V I S T A

File Manager 22.2

Advanced User Manual

March 2013

Fileman/
Lab
Agile
Project

T3 TigerTech
Trusted Talented Tireless

vista
Expertise Network

Revision History

Date	Description	Language	Authors
March 2013	Version 22.2 release	English (US)	Donald Creaven, Kathy Ice, and Frederick D.S. Marshall, based on the work of Gary Beuschel, Susan Strack, Jack Schram, Skip Ormsby, Tami Winn, Michael Ogi, and Thom Blom

Contents

Orientation

The typical end user in a VISTA system interacts with Fileman indirectly, through VISTA's clinical and administrative packages. There is no need, therefore, for a typical "user manual" for Fileman. Instead, this manual is written for advanced users of a VISTA system: the super-users and front-line programmers who can use Fileman's tools and features to customize VISTA for their facility or their department.

This manual assumes that Fileman 22.2 is already installed and running on your system. For instructions on how to install Fileman 22.2, please refer to the Fileman 22.2 Installation Guide.

This manual uses VA conventions for displaying mockups of sensitive data in illustrations and screen captures. The first three digits (prefix) of any social security numbers (SSN) will begin with "000." Patient and user names will be formatted as FMPATIENT,[N] and FMUSER,[N] respectively, where "N" represents the first name as a number spelled out.

Screen captures and computer source code are shown in a non-proportional font and enclosed within a box. The user's responses to online prompts will be boldface.

Most screen captures are shown in scrolling mode. This is because scrolling-mode prompts and responses are easier to follow on paper or in PDF form. Most of what is shown in scrolling mode can also be done in screen mode, list mode, or with a GUI; it will just look a little different.

Before You Begin

To fully understand the material in this manual, you should be reasonably familiar with the material covered in the *Fileman 22.2 Getting Started Manual*. Specifically, you should know:

- How to use the “Inquire to File Entries,” “Print File Entries,” “Search File Entries,” and “Enter or Edit File Entries” options;
- How to create, save, and use Inquire templates, Print templates, Sort templates, Search templates, and Edit templates; and
- The basic definition and use of sort qualifiers, print qualifiers, and edit qualifiers.

In order to practice the operations described in this manual, you should have super-user access to Fileman, and access to at least one file in your system.

This Manual’s Structure

This manual begins with a section called “Getting Started II.” In this section, we build directly on information you learned in the *Getting Started Manual*. This includes new uses for the options you already know, plus a few new options you can try.

The second section, “Introducing the Data Dictionary,” shows you how to view data dictionary listings for various files, and how to use that information to make your templates and other tools even more powerful.

In the third section, “Let’s Make Some Files,” you will learn how to create files, add fields, define keys and identifiers, and set up cross-references and indexes. (You’ll also learn what all those terms mean!)

In the last section, “Moving Data Around,” you will learn how to import, export, transfer, extract, and archive data.

Related Manuals and Other References

The Advanced User Manual is one part of the Fileman 22.2 documentation suite. The other manuals are:

Fileman 22.2 Release Notes
Fileman 22.2 Install Guide
Fileman 22.2 Getting Started Manual
Fileman 22.2 Security and Privacy Manual
Fileman 22.2 Programmers Manual
Fileman 22.2 Technical Manual

The other manuals are available for download at www.osehra.org.

Introduction

What is Fileman?

In discussing what Fileman *is*, it is probably best to begin with what Fileman *isn't*. Fileman isn't a clinical package like Laboratory or Pharmacy, and it isn't an administrative package like IFCAP. Fileman is an infrastructure package, meaning it is part of what makes VISTA run. Programmers use the features in Fileman when creating those other packages: Lab and Pharmacy and IFCAP and so on. Because the same infrastructure packages and tools are used throughout VISTA, similar commands, keystrokes, and shortcuts can be used in all of them. In learning to use the features in Fileman, you are learning to use features in all of the other packages.

"Fileman" is the shorter, more informal name of the package; File Manager is its formal name. As you may have guessed from the name, the specific part of the infrastructure that File Manager handles is files. That is, Fileman manages VISTA's massive database. Fileman commands are all about data: entering it, editing it, retrieving it, sorting it, printing it, making it into reports.

Although Fileman was written for VISTA, it does not need to be used with VISTA. It can be installed as a standalone database management system.

Getting Started II

Chapter 1: Templates Revisited

The Template Edit option available on the Fileman Utility Functions menu [DIUTILITY], is used to edit each of the three types of Fileman templates:

INPUT
PRINT
SORT

For each template type, a *two*-screen Screenman form is used. This allows you to edit templates in Screen Mode.

The ***first*** screen of the pair allows you to change the access privileges of the template you are editing:

READ ACCESS—This access controls which class of users [i.e., DUZ(0)] get to *use* the template.

WRITE ACCESS—This access controls which class of users gets to *change* the template.

The ***first*** screen also allows you to enter a DESCRIPTION for the purpose of documenting what the template does. This DESCRIPTION will be printed on a "TEMPLATES ONLY" data dictionary list, and in the "TEMPLATES" section of other data dictionary listings.

The ***second*** screen allows you to edit the *contents* of a template. In order to "jump" to the second screen from the first screen in a Screen Mode, you need only press the **<F1><ArrowDown>** from wherever you are on the current screen.

NOTE: The ***first*** screen provides the usual kind of field-by-field help in response to entering a single question mark ("**?**"); all help messages are displayed in the lower portion of the screen. Also, entering **<F1>H** will provide general Screenman help.

The ***second*** screen, however, does *not* provide help on individual entries. Thus, if you are building a complicated new template from scratch, it is still a good idea to use the traditional, interactive Scrolling Mode with the Enter or Edit File Entries and Print File Entries options.

Here is an example the ***first*** screen of a PRINT template using the Template Edit option:

```
Select Utility Functions Option: TEMPLATE Edit

MODIFY WHAT FILE: NEW PERSON// <Enter>
Select TEMPLATE File: PRINT template

Select PRINT TEMPLATE: XUFILEINQ
     1   XUFILEINQ
     2   XUFILEINQHDR
CHOOSE 1-2: 1 <Enter>  XUFILEINQ
Do you want to use the screen-mode version? YES// <Enter>
```

You will then be taken into a Screenman form where you can edit the template properties, as shown below:

```
TEMPLATE NAME: XUFILEINQ                          TEMPLATE TYPE:
                   (Compiled as '^XUFILEO' routine)

       DATE LAST MODIFIED: NOV 4,2004@11:29
           DATE LAST USED: MAY 17,2012
              READ ACCESS: @
             WRITE ACCESS: @
                   USER #:

            DESCRIPTION...

   HEADER:
   [XUFILEINQHDR]
    SUB-HEADER SUPPRESSED:
```

```
                   (Print Fields on Next Page...)


______________________________________________________________
Exit     Save     Next Page     Refresh

Enter a command or '^' followed by a caption to jump to a
specific field.


COMMAND: NEXT                       Press <F1>H for help    Insert
```

The dates shown following the "DATE LAST MODIFIED" and "DATE LAST USED" prompts are for informational purposes only and are *not* editable. Also, if a template has been "compiled" into a set of routines, an informational message will be displayed near the top of the screen (e.g., "Compiled as '^XUFILE0 routine").

On the ***second*** screen of the form, you will see the SORT, PRINT, or INPUT fields themselves. Thus, you can use this second screen to edit the specific template fields.

Here is an example of the ***second*** screen of a PRINT template using the Template Edit option:

```
Editing Print Template "XUFILEINQ"
============[ INSERT ]=============< (File 200)
>============[ <F1>H=Help ]====
$S(#3="@":"Programmer Access to All Files",1:"");C38;L35;""
ACCESSIBLE FILE
   NUMBER;C1;L10;"FILE#"
   ACCESSIBLE FILE;C12;L25
   DATA DICTIONARY ACCESS;R3;"DD"
   DELETE ACCESS;R5;"DELETE"
   LAYGO ACCESS;R5;"LAYGO"
   READ ACCESS;R4;"READ"
   WRITE ACCESS;R5;"WRITE"
   AUDIT ACCESS;R5;"AUDIT"


<=======T=======T=======T=======T=======T=======T=======T=======
```

As you can see from this example, fields under a Multiple field (e.g., ACCESSIBLE FILE) are *indented*. As you edit, add, and delete subfields here, you *must preserve the indentation*. The same holds true for Relational Navigation within the template; fields jumped to are in a different file and are indented an extra three spaces each. You do not have to indent each new level exactly three spaces, however, there *must* be some extra number of spaces. Then, if necessary, "un-indent" the same number of spaces to get back to a previous level.

If a SORT template has a user number (i.e., USER #), only that user can use that SORT template in the Fileman Print File Entries option. To remove this restriction, simply delete the user number by entering an at-sign ("**@**") at the "USER #" prompt.

For SORT templates, you can also use the ***first*** screen of the Template Edit option to associate a particular PRINT template with a SORT template. Thus, whenever that SORT template is invoked in the Print File Entries option, the associated PRINT template will be used by default, with *no* "FIRST PRINT FIELD:" prompt being displayed to the user.

```
Select Utility Functions Option: TEMPLATE Edit

 MODIFY WHAT FILE: NEW PERSON// <Enter>
 Select TEMPLATE File: SORT template

 Select SORT TEMPLATE: XUUFAA
Do you want to use the screen-mode version? YES// <Enter>
```

You will then be taken into a Screenman form where you can edit the properties of the template, as shown below:

```
TEMPLATE NAME: XUUFAA


       DATE LAST MODIFIED: JUL 1,2007
           DATE LAST USED: JAN 29,2013
              READ ACCESS: #
             WRITE ACCESS: #
                   USER #: [redacted]

            DESCRIPTION...
           PRINT TEMPLATE: XUUFAA



                   (Sort Fields on Next Page...)

________________________________________________________________
Exit     Save     Next Page     Refresh

Enter a command or '^' followed by a caption to jump to a
specific field.


COMMAND: NEXT                      Press <F1>H for help    Insert
```

Editing SORT template fields is particularly tricky; however, most SORT templates have only three or so sort levels.

Here is an example of the ***second*** screen of a SORT template using the Template Edit option:

```
Editing Sort Template "XUUFAA"
============[ INSERT ]=============< (File 200)
>============[ <F1>H=Help ]====
SORT BY: DATE/TIME OF ATTEMPT
From: JAN 1 1999
To: T_
   WITHIN DATE/TIME OF ATTEMPT, SORT BY: USER
   From:
   To:
       WITHIN USER, SORT BY: TYPE OF FAILED ATTEMPT
       From:
       To:


<=======T=======T=======T=======T=======T=======T=======T=======
```

The specifications for each successive level of sorting are indented further to the right. You can add or insert sort levels, however, each sort group of lines *must* be indented further to the right than the sort group above it. For each level of sorting, except when the sorting is on a Boolean value, there should be a "From:" line and a "To:" line. You can also have a fourth line that says "ASK" or "DON'T ASK," for sort ranges other than first-to-last. Remember to indent each line in a sort group by the same number of spaces.

Chapter 2: Statistics

How to Generate Statistics from Reports

Fileman currently offers three types of statistical processing:

- Descriptive
- Scattergram
- Histogram

In each case, to generate statistics from reports, you use a two-step process:

1. Use the Print File Entries or Search File Entries options to generate a Fileman report. Do not queue the report. The entries you select in your report are the ones on which statistics will be generated; the way you use sort and print qualifiers in the report affects the way statistics will be generated, as discussed later in this chapter.
2. Immediately after the report finishes, use the Statistics option on the Fileman menu to generate statistics.

The two-step process for each type of statistical output is described below.

NOTE: If you have statistical software on a personal computer, you might want to consider using Fileman's Export Tool as an alternative to Fileman's statistics options, especially if the statistics options described in this chapter do not provide the statistical analysis you need. With the Export Tool, you can export your data into a format your personal computer statistical software can read and use all of that software's capabilities to perform statistical analyses on Fileman data.

Descriptive Statistics

The Descriptive Statistics routine creates a summary report of the numeric information produced by the preceding print. The number of cases is always shown.

To get descriptive statistics for fields printed out in a report, you *must* associate one of the following qualifiers with fields in the print:

Qualifier	Description
#	Count, mean, standard deviation, minimum, and maximum
+	Count and mean

To obtain descriptive statistics:

1. Print a report, and use the # or + print qualifiers on one or more fields.
2. Immediately after the report completes, generate the Descriptive Statistics based on the report.

Initial Print

```
FIRST PRINT FIELD: #BUDGET
THEN PRINT FIELD: #COST
THEN PRINT FIELD: <Enter>
Heading (S/C): PATIENT STATISTICS// <Enter>
DEVICE: <Enter>  SSH VIRTUAL TERMINAL
Right Margin: 80// <Enter>

...SORRY, JUST A MOMENT PLEASE...

 .
 .           Output is generated here.
 .
```

Generating the Descriptive Statistics

```
Select OPTION: OTHER OPTIONS
Select OTHER OPTION: STATISTICS
Select STATISTICAL ROUTINE:  DES <Enter>  CRIPTIVE STATISTICS
            User: FMUSER,TWO        2:51 PM  02/15/13

                              DESCRIPTIVE STATISTICS

       N OF                 STANDARD
       CASES    MEAN        DEVIATION      MINIMUM     MAXIMUM

BUDGET  27    45845.1481  25685.8582    2589.0000    95200.0000
COST    27    45914.1111  25796.2936    259.0000     96000.0000
```

Scattergram

If you subtotal by two fields (i.e., sub-subtotal) in a sort, you can create Scattergrams for fields that were counted with **!**, +, or # in the corresponding print.

Only numeric values are charted. The Scattergram is scaled to fit your output device's row and column dimensions. Occurrences of more than nine points in a single print position are marked by an asterisk ("*").

Initial Print

```
Select Fileman Option: PRINT File Entries

OUTPUT FROM WHAT FILE: PATIENT// <Enter>
SORT BY: NAME// +WARD LOCATION
START WITH WARD LOCATION: FIRST// <Enter>
  WITHIN WARD LOCATION, SORT BY: +ROOM-BED
  START WITH ROOM-BED: FIRST// <Enter>
    WITHIN ROOM-BED, SORT BY: <Ent
FIRST PRINT FIELD: !WARD LOCATION
THEN PRINT FIELD: !ROOM-BED
THEN PRINT FIELD: <Enter>
Heading (S/C): PATIENT STATISTICS// <Enter>
DEVICE: <Enter>  SSH VIRTUAL TERMINAL
Right Margin: 80// <Enter>

...SORRY, JUST A MOMENT PLEASE...

 .
```

Subtotal by two fields ("+xxxxxxxx").

Use print qualifiers for these two fields ("!xxxxxxxx"), which will be used for the scattergram.

Output is generated here.

```
.
.

```

Generating the Scattergram

```
Select Fileman Option: **OTHER** Options


          Filegrams ...
          Audit Menu ...
          Screenman ...
          Statistics
          Fileman Management ...
          Data Export to Foreign Format ...
          Extract Data To Fileman File ...
          Import Data
          Browser

Select Other Options Option: **STAT**istics
Select STATISTICAL ROUTINE: **SCATTERGRAM**

DEVICE: HOME// **<Enter>**  SSH VIRTUAL TERMINAL
Right Margin: 80// **<Enter>**


               PATIENT STATISTICST  (TOTAL = 47)

    0                             2                             4
    +--------------+--------------+--------------+--------------+
 12+                           *                              +12
   │                                            2               │
   │                                            3               │
   │                                                            │
   │                                            2               │
  8+                                            2              +8
   │                             2                              │
   │                             3                              │
   │                                                            │
   │                             3                              │
  4+                             3                             +4
   │              2                                             │
   │              2                                             │
   │                               5                            │
   │              3                                             │
  0+                                                           +0
```

```
   +--------------+-------------+--------------+---------------+
                  1                              3
          X-AXIS:  WARD LOCATION   Y-AXIS:  ROOM-BED
```

Histogram

If you subtotal by one or more fields in a sort, you can get Histograms for the fields that are preceded by #, **!**, **&**, or + qualifiers in the corresponding print. The Histograms that you can produce depend on which print qualifier is used:

Qualifier	Description
!	Produces a Count Histogram
&	Produces a Sum Histogram
+	Produces Count, Sum, and Mean Histograms
#	Produces Count, Sum, and Mean Histograms

Here is an example of a using a subtotal in a print, and then producing a Count Histogram:

```
Select Fileman Option: PRINT File Entries

OUTPUT FROM WHAT FILE: PATIENT// SIGN-ON LOG
SORT BY: DATE/TIME// +NODE NAME
START WITH NODE NAME: FIRST// <Enter>
  WITHIN NODE NAME, SORT BY: <Enter>
FIRST PRINT FIELD: DATE/TIME
THEN PRINT FIELD: !NODE NAME
THEN PRINT FIELD: <Enter>



Heading (S/C): SIGN-ON LOG STATISTICS  Replace <Enter>
DEVICE: <Enter>  SSH VIRTUAL TERMINAL
Right Margin: 80// <Enter>

...HMMM, JUST A MOMENT PLEASE...
```

Subtotal on a field in the sort.

Use the "!" print qualifier, so you can create a Count Histogram on the NODE NAME field.

```
.
.     Output is generated here.
.


Select Fileman Option: OTHER Options


          Filegrams ...
          Audit Menu ...
          Screenman ...
          Statistics
          Fileman Management ...
          Data Export to Foreign Format ...
          Extract Data To Fileman File ...
          Import Data
          Browser

Select Other Options Option: STATistics
Select STATISTICAL ROUTINE: HISTOGRAM

DEVICE: HOME// <Enter>  SSH VIRTUAL TERMINAL
Right Margin: 80// <Enter>


                     COUNT, NODE NAME, BY NODE NAME

 ISC6A1 |*************
 ISC6A2 |******************
 ISC6A3 |***************************************
 ISC6A4 |****************************
        +-----+-----+-----+-----+-----+-----+-----+-----+-
              8    16    24    31    39    47    55    63
```

In addition to these fields accessed directly by Fileman, Kernel uses the NEW PERSON file to set up Fileman key variables. You can define additional NEW PERSON file fields to use to define these local variables as follows:

Field Name	Field #	Node;Piece	Description
Internal Entry Number	—	—	Used to set DUZ for the user. There is no defined .001 field on the NEW

Field Name	Field #	Node;Piece	Description
			PERSON file.
FILE MANAGER ACCESS CODE	3	0;4	Used to set DUZ(0) for user. A FREE TEXT field from 1 to 15 characters.
LANGUAGE	200.07	200;7	Used to set DUZ("LANG"). A POINTER TO A FILE field that points to the LANGUAGE file (#.85) identifying the user's language.
TIMED READ (# OF SECONDS)	200.1	200;10	Used to set DTIME for the user. A NUMERIC field with a value of 1 to 99999.

When these additional fields are defined, you can use them in a signon routine to set these key variables.

Of course, you may choose to place additional information about your users in the NEW PERSON file (#200). If you add other fields to your NEW PERSON file, use field numbers greater than 10,000 and use subfile numbers with at least 5 digits following the decimal place. Also, place the fields in global nodes subscripted with numbers greater than 10,000. If you numberspace your data elements in this way, you will be able to avoid conflicts, if you later install Kernel's NEW PERSON file.

Introducing the Data Dictionary

Chapter 3: Fileman File Structure

No, this isn't going to be one of those tedious "what is a file?" discussions. You already know what a file is. You also know what fields and records are, what it means to point to a file, and how a database works. Now it's time to "pop the hood" and find out what really makes Fileman files tick.

File Characteristics

Each file in the Fileman system has certain characteristics that you (and Fileman!) can use to identify it. The primary characteristics are:

- file name,
- file number,
- file description,
- data global, and
- the file's data dictionary entry.

You are already familiar with file names. You know about the PATIENT file, the STATE file, the NEW PERSON file, and so on. Each Fileman file has a name. However, these names are not guaranteed to be unique. It's true that most of the more important files have unique names, but there's no mechanism in Fileman to prevent developers in, say, the Pharmacy package from creating a new file called PATIENT, which is different from the PATIENT file you know.

So, although most users refer to files by their names, programmers tend to use the file's number when talking about it. This is because they know that the file number *is* unique. Somebody else could create a PATIENT file, but there will always be only one File 2.

We recommend that you start getting into the habit of using file numbers rather than file names when working with files. (You may already be doing this, depending on how much time you spend hanging out with programmers.) File numbers are a more reliable way to identify files, and as you begin to work with Fileman's more powerful tools, this will be important.

If you haven't spent a lot of time hanging out with programmers, you may not know the numbers for the files you use. The following table shows the file numbers of some of the more common files. If you need the file numbers for files specific to your division or packages, ask your IT department for assistance. They will probably be delighted that you intend to start using file numbers, so don't be shy about asking.

File name	File number
Data Dictionary	0
FILE	1
PATIENT	2
STATE	5
OPTION	19
HOSPITAL LOCATION	44
DRUG	50
LABORATORY TEST	60
NUTRITION PERSON	115
NEW PERSON	200
PATIENT MOVEMENT	405
PATIENT/IHS	9000001
VISIT	9000010

In addition to a name and number, each file also has a file description. This description is written by the person who created the file. Usually, that person was a programmer, although super-users can also create files.

Ideally, a file's description should state what kind of information is in the file, when it was created, why it was created, and which VISTA packages use the information in the file. Many file descriptions do contain that information, but unfortunately many do not. If you come across files whose descriptions are inadequate, don't be afraid to contact the person responsible for the file and ask for the description to be updated. Programmers often hate doing stuff like this, but descriptions need to be informative, and if they're not, then programmers really should expect to hear about it. And if the programmers don't want to hear about it anymore, well, they know what they need to do.

Each file has a designated data global, which is the place in the computer system where the file's data is stored. Files belonging to the same package have data globals that are close to one another; this helps the programmers find the information they're looking for.

Each file also has a data-dictionary entry, which describes the fields contained in the file. This is an important and complex piece of information, so we will discuss it in more detail a little later on. For now, let's take a closer look at how Fileman keeps track of files.

More About File Numbers

You already know, in general terms, how a file works. A file contains a number of records, and each record has a record number, even if the user doesn't see the record number. The record number uniquely identifies a particular record in a file. This is all pretty basic.

Here's where it gets trippy: Fileman files are themselves records in a file.

Wait, what? Yes, this is one of those strange, mind-expanding concepts they were fond of back in the 1970s when Fileman was being developed. The very first file in Fileman, File 1, is a list of all the files in Fileman. It is variously called File 1, the FILE file, and the file of files.

Each file in Fileman, then, has an entry in the FILE file. And the record number of that entry is the file's file number. For example, the PATIENT file is File 2; its record number in the FILE file is 2. The record number of the NEW PERSON file is 200. And so on.

In general, record numbers in a file are integers. There probably isn't a patient number 3.57 in the PATIENT file, for example. The FILE file is an exception, however. Fileman's developers try to keep all the files associated with a particular package or area of VISTA together. For example, all files associated with the Laboratory package have numbers that are close to one another.

You can probably see where this is heading. Programmers create files for a package, and number them (for example) 25, 26, and 27, but then they realize they need another file. All of the integers close to 25, 26, and 27 are already taken, so they call the new file 25.1, or something similar.

Files, therefore, can have non-integer file numbers, which means that the FILE file has non-integer record numbers. Actually, any Fileman file *can* have non-integer record numbers, although it's not always a good idea. We recommend using non-integer record numbers only in cases where it is necessary, or makes obvious sense.

The Data Dictionary

The data dictionary is a file in the Fileman system. It is often abbreviated as "DD," and we will often use that convention in this manual. Although it is a file, the DD has a few characteristics that make it different from all the other files.

For example, it is not listed in the FILE file. The mechanism for this is a little complex, but essentially, Fileman developers "hid" the DD from the FILE file by making it File 0.

Another unique characteristic of the data dictionary is that its entries are not, strictly speaking, records. In many ways they act more like files, although they are not files either. Essentially, DD entries are something in between a record and a file.

Okay, so what is contained in these not-quite-records-and-not-quite-files? Each entry in the data dictionary corresponds with an entry in the FILE file. The DD entry describes the fields contained in the file. Each field has its own characteristics: name, type of data, description, and so on. The complete list of one file's fields, and all the characteristics of the fields, constitute an entry in the DD.

Let's take a closer look at fields and how they are defined.

Field Characteristics

A file can have many fields. The DD entry for the file lists these fields, along with their attributes. These attributes include the field name, the data type, the field number, and a field description. (All fields *should* have descriptions—there is a designated place for them—but not all of them do.)

In addition to the normal attributes, certain fields can be given special designations in the data dictionary. In this section, we discuss:

- the .01 field,
- identifiers,
- required fields,
- required identifiers,
- indexes,and
- keys.

You already know that the .01 field in a file is the most important field in the file, as designated by the person who created the file. When you create a file (and you'll be doing this later in the manual), immediately after you name your new file, Fileman creates a .01 field for the file and calls it "name." The person who created the file can go in and change the name of the .01 field to something else, but they all start out as "name."

And when you think about it, a lot of the files you work with have a .01 field that is called “name.” This is partly laziness on the part of the file designers—just accepting the default—but it is mostly because for much of the information we look at, the name *is* the most important piece of information. Names are how we human beings refer to things. Patients, states, drugs, insurance providers—we call all of these things by names.

However, as anybody who has ever looked up a patient can tell you, the name is not always sufficient to uniquely identify a particular record. Other fields in a file can be designated as identifiers, which means pretty much what it sounds like: these fields can be used to assist in identifying the specific record the user is looking for.

Another thing that can be done with a field is making it a required field. Required fields, you will recall, are the ones you can’t skip, but you can jump out of. A field can also be a required identifier, which means it works in much the same way as the .01 field.

The DD also allows the file designer to say which fields are indexed. A Fileman file can have any number of indexes, and each index specifies one or more fields. Indexes can be lookup or non-lookup.

A lookup index allows users to select a record using a field, even if that field is not normally an identifier. Non-lookup indexes don’t affect users, but they can help in other ways. For example, a non-lookup index could be created to help speed up a sort. Non-lookup indexes are also useful in relational navigation, which you will learn about in a few chapters.

Keys are a relatively new addition to Fileman. A key is a way to uniquely identify any record in the file. It can be a single field (a key field) or several fields, considered together (a compound key). A file can even have more than one key, although if it does, one key is designated as the primary key.

Eventually, just about all Fileman files should have keys. At the moment, only some of them do, because keys are relatively new.

How "relatively?" Well, keys were first introduced to Fileman in 1999. And if that seems like a long time ago, remember that most of the files you use have been around for much longer than that. The first Fileman files were created in the 1970s, and some of them are still in use today. They do eventually need to have keys, but performing this refit will take some time.

Internal and External

It is worth noting that when Fileman shows you a value in a field, the value you see is not necessarily the value that Fileman has stored for the field. In many cases, the internal value (what Fileman stores) is different from the external value (what Fileman shows you).

For example, all date/time values are stored in Fileman as a single number. When you enter a date, Fileman converts it into a number, and stores it that way. When you look at the information, Fileman converts it from a number back into a date, and shows it to you in the format you are expecting to see.

When using some of Fileman's advanced features, it often makes a difference whether the internal and external values are the same, or whether they are different.

As we will see in chapter 6, super-users can ask Fileman for the internal value of a field, then use that internal value to perform various tasks.

An Expert Super-User

As a super-user, you are an expert on files. Even if the concepts in this advanced manual are new to you, we're betting that you've spent a fair amount of time in your organization's files. You know what's in those files, how they work, and how you can make them disgorge reports for you.

And you also know where those files fall short. You know which files are plagued by duplicate records (that you have to clean up!), which required fields shouldn't be required, which non-required fields *should* be required, and which fields really ought to be indexed because you sort on them all the time.

Since you have all this expertise, don't keep it to yourself. Share it with your IT department. Some of the changes you need are fairly simple. Some, honestly, are fairly complex—but either way, it won't hurt to ask. If your site's programmers know what you need, what kind of things could make the system work better for the users, then they're in a better position to do something about it. Even if they can't quite do the exact thing that you want, they may be able to do *something* to improve the situation.

But you won't know until you ask.

Record Numbers

Every record in a Fileman file has a record number, sometimes called an Internal Entry Number or IEN. The record number is created and assigned automatically each time a new record is added to the file.

A lot of users, once they hear about the record number, picture it as another field, one they never see. They think of each record in the file having a "record number" field, which is automatically filled in by Fileman. This is a perfectly adequate level of understanding for the average user, but it is not correct. And as a super-user, you need to know how it really works.

The record number is not a field; it is part of the structure of the file. A particular record number is associated with a record, but it is not part of the record. You might think of record numbers as hooks on a coat rack. You can hang a coat on a hook, and the coat is associated with the hook, but the hook is not part of the coat. Similarly, a record number is associated with a record, but is not part of it.

The record number is a guaranteed method for uniquely identifying a particular record. And until keys become more widely used, it is the only guaranteed method that will work for any file. Therefore, when Fileman chooses a record for some internal process, it always chooses by record number, never by name or anything else. Similarly, programmers writing routines for Fileman always use the record number to select a record.

So, why don't we have users select records by numbers? Wouldn't that be more reliable?

Not really. In most cases, record numbers are meaningless and arbitrary. The first record entered is record 1, the second is record 2, and so on. Forcing users to choose records based on what order somebody typed them in is just mean. More importantly, it would actually lead to more errors. What nurse is going to remember that his patient is number 12391 in the PATIENT file? He wants to select his patient by name, with other identifiers, such as birthdate and ID number, to help him get the right patient.

Far from forcing users to select records by their number, Fileman is actually set up to hide record numbers from users. One of the fundamental principles of VISTA is that you should *never* make a user pick a record using an artificial key, such as the record number, when there are better methods available. And there are almost always better methods available.

The .001 Field

There are times when the record number might be useful to users. We'll talk about a couple of examples in a bit. If a file designer feels that the record number is something the users might want to see, they can include a .001 field. Not all files have a .001 field—in fact, most of them do not. The purpose of the .001 field is to expose the record number to the user.

Although it is called a "field," the .001 is probably best imagined as a window. Remember that the record number is part of the file's structure. The .001 field, then, is a little window through which you can see a small part of the file's structure—the part that contains the record number.

An important consequence of this is that the value that appears in the .001

field cannot be deleted or changed. And by now you know why—the value that appears in the .001 field isn't like values in other fields. It isn't really "in" the field; it's part of the file's structure and Fileman won't let you mess around with it. Even if you have the level of permission that lets you change the file structure, Fileman won't let you do it at the .001 field.

When a file designer creates a .001 field, Fileman automatically calls the field "number." The file designer can rename the field later, but they usually do not. "Number" is a pretty good name for this field in most cases.

When you are using Fileman's tools to create custom reports, you can choose to print the record number as one of the fields. You can even choose to print the record number in a file that does not have a .001 field! Enter "number" as the name of a field you would like to print, and, unless the file has another field called "number," Fileman will print the record number. Even in a file that does not have a .001 field, Fileman knows what "number" should mean.

So, what files have a .001 field? Many of them do, but one example is file 50 (the DRUG file). Users can look up drugs by entering their record number.

That seems counter-intuitive at first; wouldn't most users prefer to look up drugs by their name? Most human users would. However, file 50 is also used by barcode medication administration or BCMA. BCMA is increasingly popular because it allows clinicians to administer drugs with a much lower error rate.

And what is a barcode? It's a number. Basically, barcodes are really, really reliable ways of entering a long number into a computer. But that number needs to mean something to the computer. Like, for example, the record number of a type of medication. Therefore, file 50 has a .001 field to accommodate BCMA.

Did You Just Say "Dye-Numbing?"

The .001 field is a way of exposing the record number to the user. This is done if the record number is somehow useful to the user in finding the record they need.

Sometimes the record number is useful because the file designer made it that way. It is possible to set up a file in such a way that the record number is derived from the value in the .01 field.

For example, IHS collects more information on patients than is contained in the PATIENT file. They have, in effect, a supplementary PATIENT file that holds this additional information. The record number in this supplementary file is derived from the .01 field, so that it matches the record number in the PATIENT file.

To do this, to use the .01 field to calculate the record number, file designers use a special variable called DINUM. When programmers create a file and use DINUM to calculate the record number, they call it “DINUMming,” and say that the record number has been “DINUMmed.”

If a file has a DINUMmed .01 field, Fileman will not let you change or delete its value, even if you have write or delete access to the file. The reason for this is that in this type of file, the value of the .01 field is linked to the record number. Messing with one means messing with the other, and Fileman doesn’t want people messing around with record numbers.

Pointers and Variable Pointers

You know what pointers and variable pointers are. And, by this time, you probably have a pretty good idea that pointers and variable pointers are more than just data types. They are links that establish relationships between files, between records, among all the data in Fileman’s database. Pointers and variable pointers are the things that *make* Fileman a database, rather than just a big blob of information.

More than that, pointers and variable pointers create pathways, which you can use to navigate through files. You’ll learn more about this in chapter 5.

Pointers (and variable pointers) are a good example of the kind of data whose internal value is different from its external value. Internally, pointers are stored as record numbers. Externally, they show up as the value of the 01 field. This is important information to have if you ever need to deal with dangling pointers. And you probably will.

Dangling Pointers

As you can imagine, it's not usually a good idea to move or delete a pointed-to record. It tends to break other records in other files. A pointer (or variable pointer) that no longer points to anything is called a dangling pointer. And dangling pointers are trouble. They are a form of database corruption.

You can spot a dangling pointer because, instead of showing the .01 field, it shows the record number. For example, if someone foolishly eliminated Alaska from the STATE file, then every address pointing to "Alaska" would instead show the name of the state as "2."

If you ever encounter a dangling pointer, it needs to be fixed as soon as possible. If you do not have the right permissions (or the know-how) to fix it yourself, then enlist the help of the IT department. Don't let it go. It's causing problems, even if you can't see them all.

Although you may need the IT department's help in fixing a dangling pointer, it's rare that the IT person can fix it all on their own. Figuring out where the pointer is *supposed* to be pointing requires knowledge of the subject matter. And it's likely that a programmer will not have that knowledge.

If you run into a situation where one of the files you use is having an ongoing problem with dangling pointers cropping up, you can ask your IT person to turn on auditing for the field or fields involved. When auditing is turned on, Fileman tracks what got changed, when the change was made, and who changed it. This will help you find out what keeps going wrong.

Because dangling pointers are a problem, Fileman has features to help prevent them. If a user (or super-user) attempts to move or change a pointed-to record, Fileman displays a warning and asks the person whether they are sure they want to proceed. The person can go ahead and move or change the record anyway, but at least they've been warned. Therefore, if you ever find somebody who has moved or altered pointed-to records, don't let them get away with "I didn't know." They knew. They had to know; Fileman would have told them.

Variable Pointers

In many ways, variable pointers are like standard pointers. However, they have some unique characteristics that are worth a brief discussion.

Variable pointers point to more than one file, and can select a record from any of them. When searching for a record at a variable-pointer prompt, a user can specify which file they want Fileman to look in, but they don't have to. If no file is specified, Fileman checks each of them in turn, looking for the string the user entered.

To specify a file, the user enters the file abbreviation, set up for the variable-pointer field, followed by a period, followed by the search string. For example, if a variable pointer pointed to the PATIENT file (among others), a user might enter **P.FMUSER** to search for a specific record in that file. In this example, the "P" abbreviation is a property of the variable-pointer field, not of the PATIENT file. The PATIENT file doesn't know that it is "P." A different variable-pointer field might have a different way of specifying the PATIENT file (for example, PT).

The file abbreviation can also be used to get help on that specific file. The user can, of course, obtain general help at the prompt by typing a question mark. They can obtain help on a specific file by typing the abbreviation, followed by the period and question mark (for example: **P.?**).

If a user has LAYGO access, they can add a record at a variable-pointer prompt. When adding a record, however, the user *must* specify which file it goes in. It wouldn’t be fair to make Fileman guess; Fileman is awfully smart, but in the end it’s just a computer program.

As we mentioned earlier, pointers and variable pointers have different internal and external values. For pointers, the internal value is the record number. For variable pointers, the internal value is the record number paired with an indicator for the file. This indicator is a value called the global root, which you will learn about later on.

Chapter 4: Data Dictionary Utilities

The structures of Fileman files are stored in the data dictionary (DD). There, you can find the specifications of every field in every file. Frequently, you need to know the information in the DD (usually field names and descriptions) to successfully access and use the data in Fileman's files.

The Data Dictionary Utilities submenu contains the following utilities that show information about files:

- List File Attributes
- Map Pointer Relations
- Check/Fix DD Structure
- Find Pointers Into a File

List File Attributes Option

To get a listing of the fields in a file (and other file attributes), use the List File Attributes option. This listing displays the structure of the file and the characteristics of the fields in the file; it does not show entries, records, or any data contained in the file. This information can be very useful when deciding what fields to include in a report, or what fields to edit.

You have your choice of the following formats for the listing:

- Brief
- Condensed
- Standard (or Modified Standard)
- Custom-Tailored
- Templates Only
- Global Map
- Indexes and Cross-References Only
- Keys Only

First, choose the file to display information about; you can use either its file number or name.

When you select the file, you have the option of requesting a range of files. If you select a range of files, the file number of the "go to" file *must* be higher than the file number of the "start with" file.

If you are prompted "Select SUB-FILE:", this indicates that the file you are working with has Subfiles. If you want information only about a Subfile, specify the Subfile at this prompt. If you do not choose a Subfile, your listing usually will include information about all fields in the file, including those in all Subfiles.

Next, choose a format for the listing:

```
Select LISTING FORMAT: STANDARD//  ?
  ANSWER WITH LISTING FORMAT NUMBER, OR NAME
CHOOSE FROM:
    1        STANDARD
    2        BRIEF
    3        CUSTOM-TAILORED
    4        MODIFIED STANDARD
    5        TEMPLATES ONLY
    6        GLOBAL MAP
    7        CONDENSED
    8        INDEXES AND CROSS-REFERENCES ONLY
    9        KEYS ONLY
```

Brief Data Dictionary

When you choose Brief as the data dictionary format, a brief listing will be produced; the Brief format is more readable but less complete than the default of a Standard listing. Next, you are asked for a destination for the listing's output at the "DEVICE:" prompt. You can specify any valid printer or press the **Enter** key to send output to your screen as illustrated below:

```
Select LISTING FORMAT:  STANDARD// BRIEF
ALPHABETICALLY BY LABEL?  NO// <Enter>

DEVICE: <Enter>
```

Here is a sample of a Brief data dictionary listing of an elementary file of patients:

```
BRIEF DATA DICTIONARY #16026 -- PATIENT FILE    05/31/91  PAGE 1
SITE: KDEMO V7   UCI: VAH,KXX
----------------------------------------------------------------

NAME                          16026,.01    FREE TEXT

                                  Answer must be 3-30
characters in length.

SEX                           16026,1    SET
                                         'm' FOR MALE;
                                         'f' FOR FEMALE;


DATE OF BIRTH                 16026,2    DATE
                                       TYPE A DATE BETWEEN
1/1/1860 AND 1963

RELIGION                      16026,3    POINTER TO RELIGION
FILE (#13)

DIAGNOSIS                     16026,4  16026.04
   Multiple

  DIAGNOSIS                   16026.04,.01    FREE TEXT

                                  Answer must be 3-30
characters in length.

  AGE AT ONSET                16026.04,1    NUMBER

                              Type a Number between 0 and 100, 0
Decimal Digits

  HISTORY                     16026.04,2  16026.42  WORD-
PROCESSING

PROVIDER                      16026,5    VARIABLE POINTER
              FILE  ORDER  PREFIX    LAYGO  MESSAGE
               6      1    S           n    STAFF PROVIDER
               16     2    O           y    OTHER PROVIDER

SSN                           16026,6    FREE TEXT
  Social Security Number               Enter 9 numbers without
```

```
dashes.
```

The information in the data dictionary reports originated in the definition of the file and its fields.

REF: For a detailed explanation of the source of the information displayed by the List File Attributes option, see the "Creating Files and Fields" section.

This data dictionary listing tells you that for each patient, the following information may be available:

- A NAME that is from 3 to 30 characters long.
- A recorded SEX of either **m** (MALE) or **f** (FEMALE).
- A DATE OF BIRTH.
- A RELIGION (for a list of all valid religions, we would have to consult a RELIGION file).
- One or more diagnoses and for each DIAGNOSIS; DIAGNOSIS is a Multiple-valued field that has the following information:
- An AGE AT ONSET
- A HISTORY
- A PROVIDER (e.g., a primary care physician). For a list of valid PROVIDERs, you would consult File #6 and then File #16. If the PROVIDER's name does not appear, it can then be entered in File #16, since LAYGO (Learn-As-You-Go) has been allowed.
- A SOCIAL SECURITY NUMBER (SSN).

Condensed Data Dictionary

Another format for listing a file's attributes is the Condensed format, and the following example illustrates a Condensed data dictionary listing:

```
CONDENSED DATA DICTIONARY---PATIENT FILE      (#16026)UCI:
VAH,KXX

STORED IN: ^DIZ(16026,
05/31/91    PAGE 1
----------------------------------------------------------------
```

```
----------------
                                                  FILE SECURITY
                                   DD SECURITY    : #     DELETE
SECURITY: #
                                   READ SECURITY  : #     LAYGO
SECURITY : #
                                   WRITE SECURITY : #
CROSS REFERENCED BY:
      NAME(B)


                                 FILE STRUCTURE

FIELD     FIELD
NUMBER    NAME

.01       NAME (RF), [0;1]
1         SEX (RS), [0;2]
2         DATE OF BIRTH (RD), [0;3]
3         RELIGION (P13'), [0;4]
4         DIAGNOSIS (Multiple-16026.04), [1;0]
          .01  DIAGNOSIS (MF), [0;1]
          1    AGE AT ONSET (NJ3,0), [0;2]
          2    HISTORY (Multiple-16026.42), [1;0]
               .01  HISTORY (W), [0;1]
5         PROVIDER (V), [2;1]
6         SSN (RFa), [2;2]
```

The codes in parentheses following the field names in the Condensed data dictionary contain information regarding the specifications of the field.

Here is a complete list of those codes and their meanings:

Code	Description
a	The field has been marked for auditing all the time.
e	The auditing is only on edit or delete.
A	For Multiples, new subentries can be added without being asked.
BC	The data is Boolean computed (true or false) and C the data is computed.

Code	Description
Cm	The data is Multiline computed.
D	The data is date-valued.
DC	The data is date-valued, computed.
F	The data is FREE TEXT.
I	The data is uneditable.
Jn	To specify a print length of "n" characters.
Jn,d	To specify printing "n" characters with "d" decimals.
K	The data is M code.
M	For Multiples, the user is asked for another subentry.
N	The data is NUMERIC-valued.
O	The field has an OUTPUT transform.
Pn	The data is a POINTER TO A FILE reference to file "n".
Pn'	LAYGO to the pointed-to file is not allowed.
R	Entry of data is required.
S	The data is from a discreet SET OF CODES.
V	The data is a VARIABLE-POINTER.
W	The data is WORD-PROCESSING.
WL	The WORD-PROCESSING data is normally printed in line mode (i.e., without word wrap).
X	Editing is not allowed under the Modify File Attributes option, because the INPUT transform has been modified under the Utility Functions menu [DIUTILITY].

Code	Description
*	There is a screen associated with a DATA TYPE field value of any of the following: POINTER TO A FILE VARIABLE-POINTER SET OF CODES

For example, the SSN field is required (**R**), is FREE TEXT (**F**), and is audited (**a**).

Standard and Modified Standard Data Dictionaries

The most complete information about a file is obtained by using the Standard data dictionary format, which is the default for the List File Attributes option. In addition to detailed information about every field in the file, the Standard data dictionary format gives the file access, identifiers, cross-references, other files pointing to the file, files pointed to by the file, and any templates (including forms and blocks) associated with the file.

Standard Format

Here is a sample data dictionary in Standard format:

```
STANDARD DATA DICTIONARY #16026 -- PATIENT FILE
05/31/91  PAGE 1
STORED IN ^DIZ(16026,  (1 ENTRY)   SITE: KDEMO V7   UCI: VAH,KXX

DATA          NAME                    GLOBAL        DATA
ELEMENT       TITLE                   LOCATION      TYPE
-------------------------------------------------------------------
---------------
A sample file containing some of the fields found in a file of
patient
information in a hospital database.


              DD ACCESS: #
              RD ACCESS: #
              WR ACCESS: #
```

```
             DEL ACCESS: #
           LAYGO ACCESS: #
           AUDIT ACCESS: #


CROSS
REFERENCED BY: NAME(B)


   CREATED ON: MAR 22,1991


16026,.01     NAME                      0;1 FREE TEXT (Required)

              INPUT TRANSFORM:  K:$L(X)>30!($L(X)<3)!'(X'?1P.E)
X
              LAST EDITED:      MAR 29, 1991
              HELP-PROMPT:      Answer must be 3-30 characters
in length.
              GROUP:            DEMOG
              CROSS-REFERENCE:  16026^B
                                1)= S ^DIZ(16026,"B",
$E(X,1,30),DA)=""
                                2)= K ^DIZ(16026,"B",
$E(X,1,30),DA)
                                Automatically created regular x-
ref used to
                                look-up and sort entries based
on the value in
                                the .01 (NAME) field.


16026,1       SEX                       0;2 SET (Required)

                                'm' FOR MALE;
                                'f' FOR FEMALE;
              LAST EDITED:      MAR 22, 1991
              GROUP:            DEMOG

16026,2       DATE OF BIRTH             0;3 DATE (Required)

              INPUT TRANSFORM:  S %DT="E" D ^%DT S X=Y
K:2630000<X!(1600101>X)
                                X
              LAST EDITED:      MAR 22, 1991
              HELP-PROMPT:      TYPE A DATE BETWEEN 1/1/1860 AND
1963
              GROUP:            DEMOG
```

```
16026,3       RELIGION               0;4 POINTER TO RELIGION
FILE (#13)

              LAST EDITED:      MAR 22, 1991

16026,4       DIAGNOSIS              1;0 Multiple #16026.04
                                 (Add New Entry without Asking)
STANDARD DATA DICTIONARY #16026 -- PATIENT FILE
05/31/91  PAGE 2
STORED IN ^DIZ(16026,  (1 ENTRY)   SITE: KDEMO V7   UCI: VAH,KXX

DATA          NAME                  GLOBAL        DATA
ELEMENT       TITLE                 LOCATION      TYPE
-------------------------------------------------------------------
---------------
16026.04,.01    DIAGNOSIS              0;1 FREE TEXT (Multiply
asked)

                INPUT TRANSFORM:  K:$L(X)>30!($L(X)<3) X
                LAST EDITED:      MAR 22, 1991
                HELP-PROMPT:      Answer must be 3-30 characters
in length.
                CROSS-REFERENCE:  16026.04^B
                                  1)= S ^DIZ(16026,DA(1),1,"B",
$E(X,1,30),DA)=""
                                  2)= K ^DIZ(16026,DA(1),1,"B",
$E(X,1,30),DA)


16026.04,1      AGE AT ONSET           0;2 NUMBER

                INPUT TRANSFORM:  K:+X'=X!(X>100)!(X<0)!
(X?.E1"."1N.N) X
                LAST EDITED:      APR 29, 1991
                HELP-PROMPT:      Type a Number between 0 and
100, 0 Decimal
                                  Digits

16026.04,2      HISTORY                1;0   WORD-PROCESSING
#16026.42

16026,5       PROVIDER               2;1          VARIABLE
POINTER

              FILE  ORDER  PREFIX    LAYGO  MESSAGE
                 6     1    S            n   STAFF PROVIDER
                16     2    O            y   OTHER PROVIDER
              LAST EDITED:      MAR 22, 1991

16026,6       SSN                         2;2 FREE TEXT (Required)
(audited)

              Social Security Number
              INPUT TRANSFORM:  K:$L(X)>9!($L(X)<9)!'(X?9N) X
              LAST EDITED:      MAR 22, 1991
              HELP-PROMPT:      Enter 9 numbers without dashes.
              DESCRIPTION:      An entry is required.  If you do
not know this
                                patient's Social Security
Number, enter
                                '000000000' to indicate the
number is unknown.

              GROUP:            DEMOG

      FILES POINTED TO                      FIELDS

PROVIDER (#6)                     PROVIDER (#5)

PERSON (#16)                      PROVIDER (#5)

RELIGION (#13)                    RELIGION (#3)

INPUT TEMPLATE(S):

PRINT TEMPLATE(S):
CAPTIONED                                         USER #0
ZZDIAGPRINT                 MAR 29, 1991@12:18  USER #140
   Used to print information from the DIAGNOSIS multiple.

SORT TEMPLATE(S):

FORM(S)/BLOCKS(S):
```

Modified Standard Format

Another data dictionary format is the Modified Standard format, which allows you to suppress printing the M code and to restrict the listing to specified groups of fields.

For example, the following dialogue eliminates the M code from the

Standard listing and only prints those fields in the DEMOG group (NAME, SEX, DATE OF BIRTH, and SSN):

```
Select LISTING FORMAT: STANDARD// MODified Standard
WANT THE LISTING TO INCLUDE MUMPS CODE? N// <Enter>
WANT TO RESTRICT LISTING TO CERTAIN GROUPS OF FIELDS? NO// Y
Include GROUP: DEMOG
And include GROUP: <Enter>
```

NOTE: If you answer the question concerning M code YES and do not specify any groups, the output from the Modified Standard format will be the same as that of the Standard format.

Custom-Tailored Data Dictionary

The Custom-Tailored format allows you to select attributes of the fields for your report. You decide what information will be displayed, and you determine the printed format of the output.

REF: For a detailed description of the techniques used to control the format of output, see the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

For this simple example we accept the default settings:

```
Select DATA DICTIONARY UTILITY OPTION: LIST FILE ATTRIBUTES
 START WITH WHAT FILE: PATIENT // <Enter>
      GO TO WHAT FILE: PATIENT // <Enter>
      Select SUB-FILE: <Enter>
Select LISTING FORMAT: STANDARD// CUSTOM-TAILORED
SORT BY: LABEL// ?
 ANSWER WITH ATTRIBUTE NUMBER, OR LABEL
 DO YOU WANT THE ENTIRE ATTRIBUTE LIST? Y <Enter>
```

The "SORT BY:" prompt allows you to specify the order in which the data dictionary information will be displayed.

Here, we are asking for the entire list of possible attributes about a field that can be stored in the data dictionary. Typically, no field would have a value for every one of these attributes:

```
CHOOSE FROM:
   .001        NUMBER
   .01         LABEL
   .1          TITLE
   .12         VARIABLE POINTER  (multiple)
   .2          SPECIFIER
   .23         LENGTH
   .24         DECIMAL DEFAULT
   .25         TYPE
   .26         COMPUTE ALGORITHM
   .27         SUB-FIELDS
   .28         MULTIPLE-VALUED
   .29         DEPTH OF SUB-FIELD
   .3          POINTER
   .4          GLOBAL SUBSCRIPT LOCATION
   .5          INPUT TRANSFORM
   1           CROSS-REFERENCE  (multiple)
   1.1         AUDIT
   1.2         AUDIT CONDITION
   2           OUTPUT TRANSFORM
   3           'HELP'-PROMPT
   4           XECUTABLE 'HELP'
   8           READ ACCESS (OPTIONAL)
   8.5         DELETE ACCESS (OPTIONAL)
   9           WRITE ACCESS (OPTIONAL)
   9.01        COMPUTED FIELDS USED
   10          SOURCE
   11          DESTINATION  (multiple)
   12          POINTER SCREEN
   12.1        CODE TO SET POINTER SCREEN
   12.2        EXPRESSION FOR POINTER SCREEN
   20          GROUP  (multiple)
   50          DATE FIELD LAST EDITED
   999         TRIGGERED-BY POINTER  (multiple)
 TYPE '-' IN FRONT OF NUMERIC-VALUED FIELD TO SORT FROM HI TO LO
 TYPE '+' IN FRONT OF FIELD NAME TO GET SUBTOTALS BY THAT FIELD,
  '#' TO PAGE-FEED ON EACH FIELD VALUE,  '!' TO GET RANKING
NUMBER,
  '@' TO SUPPRESS SUB-HEADER,   ']' TO FORCE SAVING SORT
TEMPLATE TYPE [TEMPLATE NAME] IN BRACKETS TO SORT BY PREVIOUS
SEARCH RESULTS

SORT BY: LABEL// <Enter>
START WITH LABEL: FIRST// <Enter>
```

```
FIRST PRINT ATTRIBUTE: ?
 ANSWER WITH ATTRIBUTE NUMBER, OR LABEL
 DO YOU WANT THE ENTIRE 36-ENTRY ATTRIBUTE LIST? N <Enter>
```

At this point, you indicate the specific attributes of the fields that you want displayed.

Here, help regarding the print formatting options will be displayed:

```
TYPE '&' IN FRONT OF FIELD NAME TO GET TOTAL FOR THAT FIELD,
  '!' TO GET COUNT, '+' TO GET TOTAL & COUNT, '#' TO GET MAX &
MIN,
  ']' TO FORCE SAVING PRINT TEMPLATE
TYPE '[TEMPLATE NAME]' IN BRACKETS TO USE AN EXISTING PRINT
TEMPLATE
YOU CAN FOLLOW FIELD NAME WITH ';' AND FORMAT SPECIFICATION(S)
```

Now, by using either the label or the corresponding number of the attribute you want, you select the information you want in your customized data dictionary listing:

```
FIRST PRINT ATTRIBUTE: LABEL
THEN PRINT ATTRIBUTE: TYPE
THEN PRINT ATTRIBUTE: DATE FIELD LAST EDITED
THEN PRINT ATTRIBUTE: <Enter>
HEADING: FIELD SEARCH// CUSTOM-TAILORED OUTPUT
DEVICE: <Enter>
```

You can save the selected attributes in a PRINT template.

REF: For details, see the "Print: How to Print Reports from Files" chapter in the *Fileman User Manual*.

The output will look like this:

```
PATIENT FILE CUSTOM-TAILORED OUTPUT       MAY 31,1991  11:10
PAGE 1
                                                 DATE FIELD
LABEL                           TYPE             LAST EDITED
----------------------------------------------------------------

```

```
DATE OF BIRTH                    DATE/TIME        MAR 22,1991
DIAGNOSIS                        FREE TEXT
NAME                             FREE TEXT        MAR 29,1991
PROVIDER                         VARIABLE-POINTER MAR 22,1991
RELIGION                         POINTER          MAR 22,1991
SEX                              SET              MAR 22,1991
SSN                              FREE TEXT        MAR 22,1991
```

NOTE: With the Custom-Tailored format, in order to get information about fields in a Multiple, you *must* specifically ask for that Multiple by entering its name at the "Select SUB-FILE:" prompt.

Templates Only Format

The Templates Only format displays information about the templates (including forms and blocks) associated with a file. The output resembles the last part of the Standard data dictionary output.

Global Map

The Global Map format shows the actual structure of the global (file) that contains the data for the file and its templates. This information is of primary interest to programmers who can control how data is stored.

Here is a sample Global Map:

```
GLOBAL MAP DATA DICTIONARY #16026 -- PATIENT FILE
05/31/91  PAGE 1
STORED IN ^DIZ(16026,  (1 ENTRY)   SITE: KDEMO V7  UCI: VAH,KXX
----------------------------------------------------------------

CROSS
REFERENCED BY: NAME(B)

^DIZ(16026,D0,0)= (#.01) NAME [1F] ^ (#1) SEX [2S] ^ (#2) DATE
OF BIRTH [3D]
               ==>^ (#3) RELIGION [4P] ^
^DIZ(16026,D0,1,0)=^16026.04A^^  (#4) DIAGNOSIS
^DIZ(16026,D0,1,D1,0)= (#.01) DIAGNOSIS [1F] ^ (#1) AGE AT ONSET
[2N] ^
^DIZ(16026,D0,1,D1,1,0)=^16026.42^^  (#2) HISTORY
^DIZ(16026,D0,1,D1,1,D2,0)= (#.01) HISTORY [1W] ^
```

```
^DIZ(16026,D0,2)= (#5) PROVIDER [1V] ^ (#6) SSN [2F] ^


INPUT TEMPLATE(S):
^DIE(30)= ZZUPDATE

PRINT TEMPLATE(S):
^DIPT(.01)= CAPTIONED
^DIPT(60)= ZZDIAGPRINT

SORT TEMPLATE(S):
```

An understanding of these data dictionary listings is the key to displaying, changing, and deleting the data in individual file entries.

Indexes and Cross-references Only

The Indexes and Cross-References Only format shows the Traditional cross-references and New-Style indexes that are defined on a file.

Here is a sample Indexes and Cross-References Only data dictionary listing:

```
INDEX AND CROSS-REFERENCE LIST -- FILE #16026   12/24/98  PAGE 1
---------------------------------------------------------------------------

File #16026

  Traditional Cross-References:

  B    REGULAR
            Field:  NAME  (16026,.01)
      Description:  Automatically created regular x-ref used to
look-up and
                    sort entries based on the value in the .01
(NAME) field.
                    1)= S ^DIZ(16026,"B",$E(X,1,30),DA)=""
                    2)= K ^DIZ(16026,"B",$E(X,1,30),DA)

  New-Style Indexes:

  KEYA (#6)    RECORD    REGULAR    IR    LOOKUP & SORTING
       Unique for:  Key A (#5), File #16026
      Short Descr:  Uniqueness Index for Key 'A' of File #16026
```

```
        Set Logic:  S ^DIZ(16026,"KEYA",X(1),X(2),DA)=""
       Kill Logic:  K ^DIZ(16026,"KEYA",X(1),X(2),DA)
       Whole Kill:  K ^DIZ(16026,"KEYA")
             X(1):  NAME  (16026,.01)  (Subscr 1)
             X(2):  SSN  (16026,6)  (Subscr 2)

Subfile #16026.04

  Traditional Cross-References:

  B    REGULAR
            Field:  DIAGNOSIS  (16026.04,.01)
                    1)= S ^DIZ(16026,DA(1),1,"B",
$E(X,1,30),DA)=""
                    2)= K ^DIZ(16026,DA(1),1,"B",$E(X,1,30),DA)
```

Keys Only

The Keys Only format shows the keys that are defined on a file.

Here is a sample Keys Only data dictionary listing:

```
KEY LIST -- FILE #16026                      12/24/98    PAGE 1
---------------------------------------------------------------


  FILE #16026
  -----------
  PRIMARY KEY:         A (#5)
    Uniqueness Index: KEYA (#6)
         File, Field: 1) NAME (16026,.01)  2) SSN (16026,6)
```

Map Pointer Relations Option

The Map Pointer Relations option on the Data Dictionary Utilities submenu creates a graphic representation of the pointer relationships between files. (Files are linked by POINTER TO A FILE and VARIABLE-POINTER field types.

REF: These field types are described in the "Creating Fields" topic in the "Creating Files and Fields" section.

You select an application package, a file, or a group of files to be mapped. If you select a package to map, you are given the opportunity to exclude a file or files from the map. The initial dialogue goes like this:

```
Select DATA DICTIONARY UTILITY OPTION: MAP POINTER RELATIONS

Prints a graph of pointer relations in a database of FileMan
files
named in the Kernel PACKAGE file (9.4) or given separately.
Works best with 132 column output!

Select PACKAGE NAME: <Enter>

Enter files to be included
Add FILE: PATIENT
Add FILE: <Enter>

Files included                    16026  PATIENT

Enter name of file group for optional graph header: PATIENT FILE

DEVICE: HOME// <Enter>
```

In this instance, only a single file, the PATIENT file, has been selected for mapping. Of course, a more useful and complex map would be produced if an entire package or a large, related group of files were mapped.

NOTE: You *must* have DD access to the PACKAGE file (#9.4) and to the files chosen at the "Add FILE:" prompt.

The output consists of three columns. The middle column has the target file or files, each surrounded by a box. This is the file or group of files that you asked to be mapped. To the left, in the first column, are the files and fields that point to the target file. An abbreviated description of the field is shown. To the right, in the last column, are any files that are pointed to by the target file. The output's heading contains brief descriptions of the codes used.

A possible output for the PATIENT file is shown below:

```
    File/Package:  PATIENT FILE              Date: MAY 31,1991
```

```
  FILE (#)                    POINTER          (#) FILE
   POINTER FIELD               TYPE           POINTER FIELD
FILE POINTED TO
-------------------------------------------------------------------
          L=Laygo      S=File not in set      N=Normal Ref.
C=Xref.
          *=Truncated      m=Multiple           v=Variable
Pointer

                                        ---------------------
   ADMISSIONS (#16999)                  |  16026   PATIENT   |
    CLIENT  ..............  (N S L)->   |   RELIGION         |->
RELIGION
                                        | v PROVIDER         |->
PERSON
                                        |                    |->
PROVIDER
                                        ---------------------
```

This output shows that the CLIENT field in the ADMISSIONS file points to the PATIENT file. LAYGO additions are allowed and the ADMISSIONS file is not in the set of files being mapped. Further, the RELIGION field in the PATIENT file points to the RELIGION file and the PROVIDER field, a "**v**" (VARIABLE-POINTER), points to the PERSON and PROVIDER files. If the target file points to a file that is not in the account, the map shows:

```
*** NONEXISTENT FILE ***
```

REF: For a more elaborate example of a pointer map, see the "Pointer Map" section in the *Fileman Technical Manual*.

Check/Fix DD Structure Option

In order to ensure that the internal structure of your files and subfiles is consistent, use the Check/Fix DD Structure option from the Data Dictionary Utilities submenu. You *must* have READ access to the files being analyzed. In addition, you need DD access for this option to correct erroneous nodes.

This utility looks at a file's identifiers, cross-references, POINTER TO A FILE, VARIABLE-POINTER, and COMPUTED fields. If there are inconsistencies or conflicts between the information in the data dictionary and the structure of the file's global nodes, the Check/Fix DD Structure option will note them.

If you want, the Check/Fix DD Structure option will correct inconsistencies found in the data dictionary. The process will *not* change any file structures; it only removes or corrects unnecessary or incorrect DD nodes. Data is *not* affected.

The dialogue for running this option is simple. You specify the file or files you want to check and indicate whether you want to delete incorrect nodes. Then the progress of the checking is displayed followed by a report of any discrepancies found or any changes made. For example:

```
Select DATA DICTIONARY UTILITY OPTION: CHECK/FIX DD STRUCTURE

Check the Data Dictionary.

 START WITH WHAT FILE: 16033
      GO TO WHAT FILE: <Enter>
Remove erroneous nodes? NO// YES
DEVICE: HOME// <Enter>  DECSERVER

Checking file # 16033
     Checking 'ID' nodes for 'Q'.
     Checking 'IX' nodes.
     Checking 'PT' nodes.
     File: 16037 Field: .01 is not a pointer.
          ^DD(16033,0,"PT",16037,.01) was killed.
     Checking FIELDs....
   Checking subfile # 16033.04
     Checking 'IX' nodes.
     Checking FIELDs...
   Checking subfile # 16033.42
     Checking FIELDs.
   Returning to subfile 16033.04.
   Returning to main file.......
   Checking subfile # 16033.01
     Checking 'IX' nodes.
     Checking FIELDs.
   Returning to main file.........
```

In the previous example, the check is being run on a single file. Correction of erroneous nodes has been requested. An incorrect "PT" node was found and deleted.

NOTE: Subfiles are inspected, too.

Application developers might use this tool to clean up their files before export. Site managers may find the reporting function useful for checking a package's files after installation. Erroneous nodes that are found by this option may be remnants of prior versions of the files; the current install may not be to blame.

Find Pointers into a File Option

You can identify entries pointing to a file using the Find Pointers into a File Option. This option helps you analyze relationships between files and their entries. It is accessed from the Data Dictionary Utility menu.

After selecting the file to be analyzed, you have three choices:

- Find all of the entries in other files that point to a particular entry in your file
- Find all of the entries in other files that point to any entry in your file. This choice will also find entries in other files that point to a non-existent entry in your file (dangling pointers to your file).
- Find all entries in other files that point to a non-existent entry in your file (dangling pointers to your file).

You can send the output of this option to the device of your choosing.

The following sample dialogue shows how you can use the Find Pointers into a File Option to identify all entries in other files that point to the "German" entry in the Language file. The resulting output identifies the three files that have pointers to the German entry in the Language file. For each entry it lists the IEN, value of the .01 field, and the Field Name in the pointing file. If you asked for either all the pointers into the file or only the non-existent pointers, similar reports would be created.

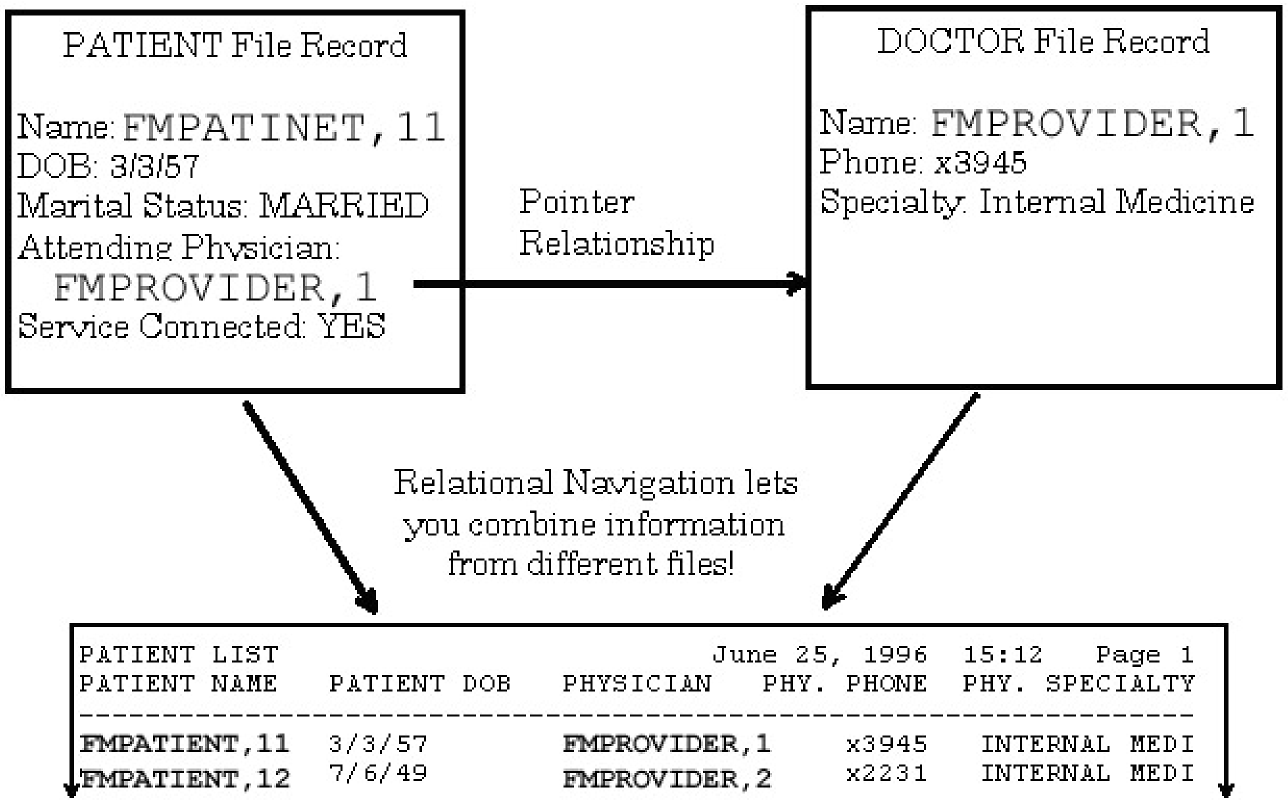
```
Select OPTION: DATA DICTIONARY UTILITIES
Select DATA DICTIONARY UTILITY OPTION: FIND POINTERS INTO A FILE

THIS UTILITY TRIES TO FIND ALL ENTRIES IN ALL FILES POINTING TO
A CERTAIN FILE


Select FILE: LANGUAGE

     Select one of the following:

          1         One particular LANGUAGE Entry
          2         All LANGUAGE Entries
          3         Non-existent LANGUAGE Entries

Find pointers to: All LANGUAGE Entries// 1  One particular
LANGUAGE Entry
Find pointers to LANGUAGE Entry: GERMAN
     1   GERMAN       DE     DEU
     2   GERMAN, LOW    LOW GERMAN          NDS
     3   GERMAN, MIDDLE HIGH (CA. 1050-    MIDDLE HIGH GERMAN
GMH
     4   GERMAN, OLD HIGH (CA. 750-1050    OLD HIGH GERMAN
GOH
     5   GERMAN, SWISS    SWISS GERMAN          GSW
CHOOSE 1-5: 1  GERMAN     DE     DEU
DEVICE: HOME// <Enter>   TELNET



***LANGUAGE: GERMAN***
FILE .84 (DIALOG)
         `820        820                        LANGUAGE
         `7003       7003                       LANGUAGE
FILE 200 (NEW PERSON)
         `31         USER,THIRTYONE             LANGUAGE
FILE 999991 (ZZD TEST FILE1)
         `2          SECOND ENTRY               LANGUAGE
POINTER
```

Chapter 5: Relational Navigation

Relational navigation gives you a way to reach beyond the current file to reference fields within other files.

Suppose, for example, you are doing a printout from the PATIENT file. In the PATIENT file, there is a pointer to the DOCTOR file. This links a given patient to a given doctor. But the only information about the doctor available from the point of view of the PATIENT file is the doctor's name. What if, in your printout, you want to print the doctor's name, phone number, and specialty (where phone number and specialty are fields in the DOCTOR file)?

The answer is to use relational navigation. By using the pointer relationship between the PATIENT and the DOCTOR file, you can start from the PATIENT file, and for each record in the PATIENT file, retrieve not only the name of the doctor for that patient, but also additional information about the doctor from the DOCTOR file.



```
PATIENT LIST                                  June 25, 1996  15:12   Page 1
PATIENT NAME   PATIENT DOB   PHYSICIAN    PHY. PHONE  PHY. SPECIALTY
--------------------------------------------------------------------
FMPATIENT,11   3/3/57        FMPROVIDER,1     x3945    INTERNAL MEDI
FMPATIENT,12   7/6/49        FMPROVIDER,2     x2231    INTERNAL MEDI
```

You can use relational navigation in many places in Fileman to *move beyond the current file* and retrieve or edit information in related files' records, including:

- Reports (Print Fields, Sort Criteria, Search Criteria)
- Editing Records (edit information in related files, not just current file)
- Computed Expressions
- COMPUTED Fields
- Within word processing **|Windows|**

The syntax to perform relational navigation, called Extended Pointer syntax, is discussed throughout this chapter.

Several types of pointer relationships between files can be exploited to combine information:

- Simple Extended Pointer (most common)
- Backward Extended Pointer
- Join Extended Pointer

A special form of relational navigation, called relational jumping, uses these pointer relationships to let you "jump" from one file to another. This makes it easier to specify a group of fields from another file when specifying what fields to edit, search, print, or sort by in interactive Fileman.

Simple Extended Pointer

The most common form of relational navigation uses *simple extended pointers*. This type of navigation requires a pointer field to exist from the current file to another file. Using a pointer field from an entry in the current file, you can easily retrieve information from the pointed-to entry in another file.

For example, suppose you are printing a report from the PATIENT file (#2). Further suppose that the PATIENT file has a pointer field called ATTENDING PHYSICIAN field to the DOCTOR file. Now, what if you wanted to include the phone number of the attending physician for each patient in your report from the PATIENT file? The attending physician's phone number is stored in the DOCTOR file, not the PATIENT file.

You can include the attending physician's phone number for each patient in your report, by using a simple extended pointer at the "PRINT FIELD:" prompt:

```
    PRINT FIELD: ATTENDING PHYSICIAN:PHONE NUMBER
```

You can use simple extended pointers in many places in Fileman, including:

- Reports (Print Fields, Sort Criteria, Search Criteria)

- Editing Records (edit information in related files, not just current file)
- Computed Expressions
- COMPUTED Fields
- Within word processing |**Windows**|

The syntax for simple extended pointers is described below.

Simple Extended Pointer Syntax (Short form)

With simple extended pointers, there *must* be an existing relationship based on a pointer field from the current file to the file you are interested in. In this case, you can reference a field in a pointed-to entry by using the following syntax:

```
pfield:element
```

"**Pfield**" is the name (or number, preceded by #) of a pointer field in the current file, and "element" is an element that exists in the field to which pfield points. This is called the short form of extended pointer syntax.

For example, since ATTENDING PHYSICIAN is a pointer field in the current file to the (fictitious) DOCTOR file, the short form of extended pointer syntax to reference the PHONE NUMBER field in the (fictitious) DOCTOR file would be:

```
ATTENDING PHYSICIAN:PHONE NUMBER
```

Simple Extended Pointer Syntax (Long Form)

The most complete or general form of extended pointer syntax (also called long form) is shown below:

```
expr:file:element
```

OR

```
expr IN file FILE:element
```

"**Expr**" is any expression that applies to the file that is your current context. "**File**" is the name of any file. "**Element**" is any element (field) in the file named by "File".

For example, since ATTENDING PHYSICIAN is a pointer field in the current file to the DOCTOR file, the long form of extended pointer syntax to reference the PHONE NUMBER field in the DOCTOR file would be:

```
ATTENDING PHYSICIAN:DOCTOR:PHONE NUMBER
```

OR

```
ATTENDING PHYSICIAN IN file DOCTOR:PHONE NUMBER
```

Relational Query Example

You can use simple extended pointers to make relational queries. For example, suppose you want to print all patients who are older than their attending physicians. A field in the PATIENT file called ATTENDING PHYSICIAN points to the DOCTOR file. Given a field PT AGE in the PATIENT file and a field DR AGE in the DOCTOR file, you can use the Print File Entries option and then enter the information that follows:

```
OUTPUT FROM WHAT FILE: PATIENT
SORT BY:  NAME// PT AGE> (ATTENDING PHYSICIAN:DR AGE)
WITHIN PT AGE>(ATTENDING PHYSICIAN:DR AGE), SORT BY: <Enter>

FIRST PRINT FIELD: NAME
```

Here, the simple extended pointer (ATTENDING PHYSICIAN:DR AGE) is used to make a comparison between values in fields in two different files.

COMPUTED Field Example

Suppose the PATIENT file has an ATTENDING PHYSICIAN field that points to the DOCTOR file. The DOCTOR file, in turn, has a field called SPECIALTY. If you want to create a COMPUTED field within the PATIENT file data dictionary that is equivalent to the SPECIALTY field in the DOCTOR file, you can define a COMPUTED field as:

```
   'COMPUTED-FIELD' EXPRESSION: ATTENDING PHYSICIAN:SPECIALTY
```

The file does not have to be specified in this case since there is a direct link between the two files through the pointer field. This is an example of the ***short form*** of the simple extended pointer syntax.

An equivalent computed expression, which explicitly identifies the file is: ATTENDING PHYSICIAN IN DOCTOR FILE:SPECIALTY. This is the ***long form*** of the syntax. It is "long" because the file name is included.

How to Navigate With a Variable Pointer Field

If the pointing field is a variable pointer, the long form of the extended pointer syntax *must* be used so that Fileman will know which of the pointed-to files to search. Here is the syntax:

```
vpfield IN file FILE:element
```

OR

```
vpfield:file:element
```

"**Vpfield**" is the variable-pointer field in the current file, "**file**" is one of the possible pointed-to files, and "**element**" applies to that pointed-to file.

The following is an example from the PATIENT file where the PROVIDER field is a variable pointer to either the PHYSICIAN file or the PERSON file, and PHONE is a field in the PERSON file. You could enter the following print specifications:

```
FIRST PRINT FIELD: NAME
THEN PRINT FIELD: PROVIDER
THEN PRINT FIELD: FILE(PROVIDER)
THEN PRINT FIELD: PROVIDER:PERSON:PHONE
THEN PRINT FIELD: <Enter>
```

You would receive the following output:

```
NAME         PROVIDER         FILE(PROVIDER)   PROVIDER:PERSON:PHONE
-------------------------------------------------------------------
FMPATIENT,13     FMPROVIDER,3         PHYSICIAN
FMPATIENT,14     FMPROVIDER,4         PERSON              555-3332
```

The long form simple pointer asked for the PHONE field from the PERSON file. Only the variable pointer from the FMPATIENT,14 entry pointed to the PERSON file. Thus, only his phone number is displayed.

Relational Jumps Across Files

In interactive Fileman mode, you can use the following syntax:

file:

Doing this changes your context to the file you specify; you "jump" to the specified file. You can then select fields from the file to which you have jumped. You can only do this in four places in Fileman:

1. "EDIT WHICH FIELD:" prompt (Enter or Edit File Entries option)
2. "SEARCH FOR FIELD:" prompt (Search File Entries option)
3. "SORT BY:" prompt (Print File Entries and Search File Entries option)
4. "PRINT FIELD:" prompt (Print File Entries and Search File Entries option)

Relational jumping is mainly a convenience to make it easier to select more than one field from another file—by letting you temporarily "jump" to the other file, it's easier to pick all the fields you want directly, rather than having to use extended pointer syntax to specify each field.

NOTE: When sorting, printing, searching, or editing, if you want to reference several fields from another file, it is more efficient to jump to the file and specify the needed fields than it is to use the extended pointer syntax to reference the fields one at a time. Multiple uses of the extended pointer cause multiple relational jumps.

There are three types of relational jumps that correspond to the three extended pointer syntax:

Type	Example
Simple (short form)	ATTENDING PHYSICIAN:
Simple (long form)	PROVIDER IN PERSON FILE:
Backward	RADIOLOGY EXAM:
Join	PAYSCALE IN FACTOR FILE:

Within the Enter or Edit File Entries option, for example, you can respond to the prompts as depicted in the dialogue that follows:

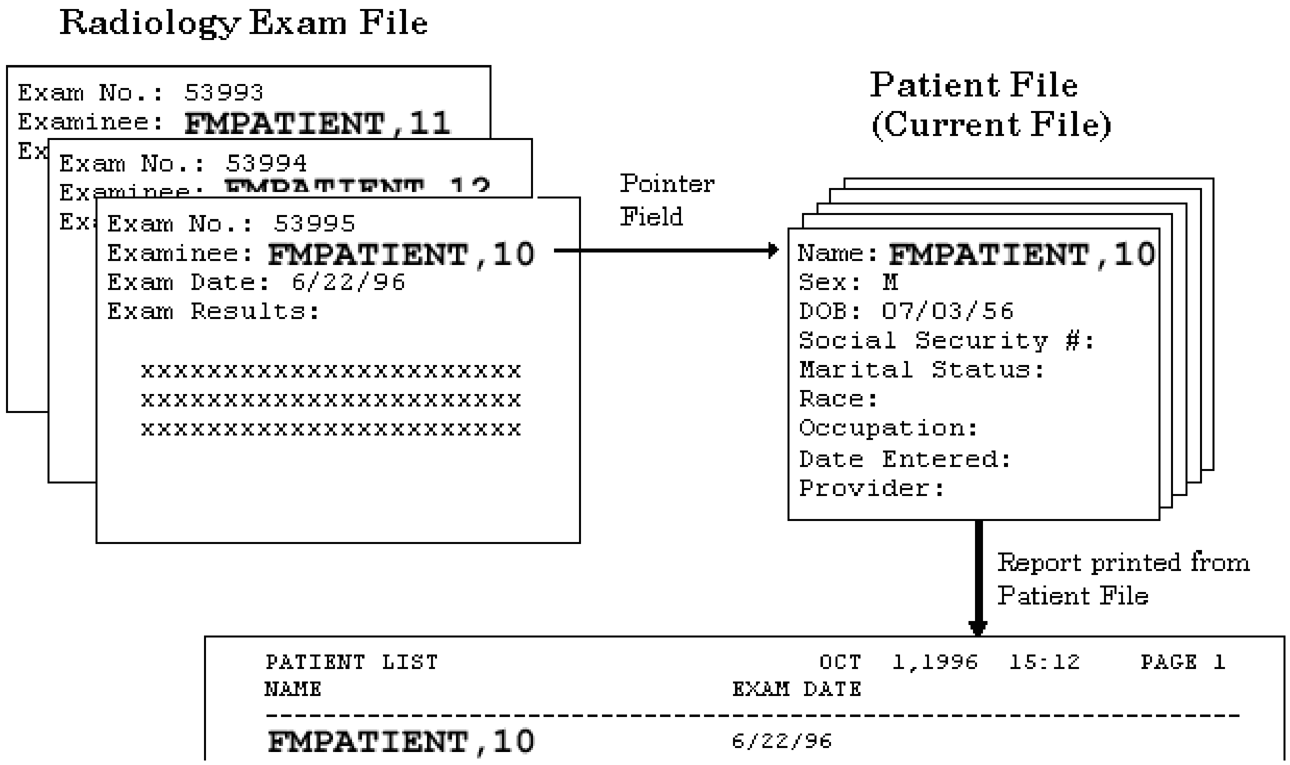
```
INPUT TO WHAT FILE: PATIENT
EDIT WHICH FIELD: ALL// NAME
THEN EDIT FIELD: ATTENDING PHYSICIAN:
EDIT WHICH DOCTOR FIELD: ALL// NAME;"PHYSICIAN NAME"
THEN EDIT DOCTOR FIELD: NICKNAME
THEN EDIT DOCTOR FIELD: <Enter>
THEN EDIT FIELD: <Enter>
```

Relational Jump!

Because of a pointer linkage between the ATTENDING PHYSICIAN field in the PATIENT file and the DOCTOR file, you can use the simple, short form of the extended pointer to navigate to the DOCTOR file. Then, during an interactive editing session you can specify the fields you want to edit for each patient. In this case, after you edit the patient's name, you can edit that patient's physician's name and nickname.

Backward Extended Pointer

Simple extended pointers let you retrieve information from an entry in another file that the current entry explicitly points to through a POINTER TO A FILE field. What if you wanted to go the other way—retrieve information from an entry in another file that points to (not from) the current entry?



```
PATIENT LIST                          OCT  1,1996  15:12    PAGE 1
NAME                                 EXAM DATE
-------------------------------------------------------------------
FMPATIENT,10                         6/22/96
```

Suppose you have selected the PATIENT file and you want to list dates of radiology exams for certain patients. If the pointer is from the RADIOLOGY EXAM file to the PATIENT file (*not* the other way around), you can list the radiology exam dates using a *Backward Extended Pointer.*

The POINTER TO A FILE field to the current file from the pointing file *must be cross-referenced*. If the POINTER TO A FILE field is located in a Subfile, then the *whole* file *must* be cross-referenced by the pointer.

To use a Backward Extended Pointer, you *must* make a relational jump from the current file to the file in question (enter the name of the file pointing to the current file, followed by a colon). Once you make the relational jump to the backwards-pointer-linked file, specify which fields/elements to access in that file.

Returning to the situation mentioned above, within the RADIOLOGY EXAM file there is a field called EXAMINEE pointing back to our PATIENT file. That EXAMINEE pointer field is cross-referenced. You want to list the EXAM DATE field from the RADIOLOGY EXAM file entries that point back to a patient. From the PATIENT file, enter:

```
FIRST PRINT FIELD: NAME;N;S1
THEN PRINT FIELD: RADIOLOGY EXAM:
   By 'RADIOLOGY EXAM', do you mean the RADIOLOGY EXAM File,
       pointing via its 'EXAMINEE' Field? YES// <Enter>
  THEN PRINT RADIOLOGY EXAM FIELD: EXAM DATE
  THEN PRINT RADIOLOGY EXAM FIELD: <Enter>
THEN PRINT FIELD: <Enter>
```

Relational Jump!

As indicated by this example, you did not have to specify the EXAMINEE field. That field was identified because it is a field in the RADIOLOGY EXAM file that points back to the current file.

The following is the output produced by these print specifications:

```
PATIENT LIST                          OCT  1,1996  15:12    PAGE 1

NAME                                 EXAM DATE
-------------------------------------------------------------------

FMPATIENT,13                         DEC 22,1995

FMPATIENT,14

FMPATIENT,15                         1995
                                     1993

FMPATIENT,10                         SEP 29,1995
                                     JUN 22,1996
```

The resulting output is a two-column report containing names from the PATIENT file and corresponding examination dates from the RADIOLOGY EXAM file. Since there may be several RADIOLOGY EXAM file entries for a given patient, this report is an example of a Multiple-valued (Multiline) result being returned.

REF: For more information on Multiline results being returned, see the "Multiline Return Values" section.

You can use Backwards Extended Pointers in the following places in Fileman:

"EDIT WHICH FIELD:" prompt (Enter or Edit File Entries option)
"SEARCH FOR FIELD:" prompt (Search File Entries option)
"SORT BY:" prompt (Print File Entries and Search File Entries option)
"PRINT FIELD:" prompt (Print File Entries and Search File Entries option)

Join Extended Pointer

You can establish an extended pointer link even if there is no pre-existing pointer relationship between the two files. You use a value from one file to do a lookup in a second file.

Suppose you store in the PAY FACTOR file a list of factors for calculating taxes. Each entry in this file corresponds to a different pay scale. In the PERSONNEL file, you have a field called PAYSCALE. You want to retrieve the value of a field DEDUCTION in the PAY FACTOR entry that equals the PAYSCALE field for each entry in the PERSONNEL file. You can create a COMPUTED field expression in the PERSONNEL file:

```
'COMPUTED-FIELD' EXPRESSION: PAYSCALE IN PAY FACTOR
FILE:DEDUCTION
```

NOTE: PAYSCALE was not defined as pointing to the PAY FACTOR file. The link to that file is made by the COMPUTED field definition. PAYSCALE could itself be a COMPUTED field. In this situation, the value of the PAYSCALE field in the PERSONNEL file is used to do a normal lookup in the PAY FACTOR file using all lookup type cross-references.

In database terminology, this extended pointer capability is similar to a **JOIN** operation because you can specify at any time a new relationship between two formerly unrelated files. Therefore, we call this type of pointing the Join Extended Pointer.

Limitations

If the join expression matches more than one entry in the file being joined, the first matching entry (by internal entry number) is returned as the result of the join. Thus, if your join expression is likely to match more than one entry, be aware that only the *first* matching entry is returned.

Example

You could find out if any entries in the PERSONNEL file could be matched against the NAME field in the PATIENT file just by specifying the following sort:

```
OUTPUT FROM WHAT FILE: PATIENT

SORT BY: NAME IN PERSONNEL FILE
```

The expression at the "SORT BY:" prompt selects entries in the PERSONNEL file where the value of the NAME field in the PATIENT file matches the PERSONNEL file's .01 field. The PATIENT file's NAME field is being used as a lookup in the PERSONNEL file. Since we are evaluating the .01 field of the PERSONNEL file, the "**:element"** part of the extended pointer syntax is unnecessary.

Multiline Return Values

When you use extended pointer syntax, a lookup is performed in the navigated-to file. This lookup usually evaluates to a single value. However, in some situations, extended pointer syntax can end up returning a Multiple-valued or "Multiline" result. Multiline responses can be generated by:

Simple Pointer to a WORD-PROCESSING Field
Simple Pointer to a Multiple
Backward Pointer

You *cannot* use extended pointer syntax that can evaluate to a Multiline value at Fileman's "SORT BY:" and "SEARCH FOR FIELD:" prompts. Some of the ways in which you *can* use extended pointers that evaluate to a Multiline value are:

As the definition of a COMPUTED field.
Within word processing |**Windows**| (so one document can call another document to print inside it).
For input to word processing data elements (so you can use the Enter or Edit File Entries option to stuff one document into another).
As the name of a transfer document in the Line Editor's Transfer option.
As a Print Field: specification in the Print File Entries option.
In an INPUT template when a multi-valued field is being edited.

WORD-PROCESSING Field

WORD-PROCESSING field names (or field numbers preceded with a `#`) are allowed as elements in extended pointer expressions. For example, in the PATIENT file the HISTORY field is in the DIAGNOSIS Multiple. You can define this computed expression:

```
    "B-12 Deficiency" IN DIAGNOSIS FILE:HISTORY
```

This Multiline computed expression would signify the WORD-PROCESSING HISTORY field text associated with a patient's B-12 Deficiency DIAGNOSIS. A lookup is done on the DIAGNOSIS Multiple using "B-12 Deficiency" as the lookup value. If the patient does not have that DIAGNOSIS (or no HISTORY is associated with it), the value of this extended pointer expression would be null.

Multiples

You can use the simple pointer syntax to get data from Multiples of files pointed to by other files. The RADIOLOGY EXAM file described above points to the PATIENT file by way of the EXAMINEE field. In the PATIENT file there is a DIAGNOSIS Multiple. You could obtain a list of diagnoses associated with RADIOLOGY EXAM file entries by doing the following:

```
Select OPTION: PRINT FILE ENTRIES

OUTPUT FROM WHAT FILE: RADIOLOGY EXAM// <Enter>
SORT BY: NAME// <Enter>
START WITH NAME: FIRST// <Enter>
FIRST PRINT FIELD: TEST NUMBER
THEN PRINT FIELD: EXAMINEE:DIAGNOSIS
THEN PRINT FIELD: <Enter>
HEADING: RADIOLOGY EXAM LIST// <Enter>
STORE PRINT LOGIC IN TEMPLATE: Exam Diagnoses
```

For each entry in the RADIOLOGY EXAM file, EXAMINEE points to an entry in the PATIENT file. The diagnoses associated with that patient are returned as the Multiline output of the expression EXAMINEE:DIAGNOSIS.

Backward Pointer

The following example shows how you can use the cross-referenced Backward Pointer that yields a Multiline response in an INPUT template:

```
INPUT TO WHAT FILE: PATIENT

EDIT WHICH FIELD: ALL// NAME
THEN EDIT FIELD: RADIOLOGY EXAM:
   By 'RADIOLOGY EXAM', do you mean the RADIOLOGY EXAM File,
       pointing via its 'EXAMINEE' Field? YES// <Enter>

WILL TERMINAL USER BE ALLOWED TO SELECT PROPER ENTRY IN
'RADIOLOGY  EXAM' FILE?  YES// <Enter>  (YES)
DO YOU WANT TO PERMIT ADDING A NEW 'RADIOLOGY EXAM' ENTRY? NO//
<Enter>
```

```
   EDIT WHICH RADIOLOGY EXAM FIELD: DATE OF EXAM
   THEN EDIT WHICH RADIOLOGY EXAM FIELD: RESULTS
   THEN EDIT WHICH RADIOLOGY EXAM FIELD: <Enter>
THEN EDIT FIELD: ATTENDING PHYSICIAN
THEN EDIT FIELD: <Enter>
STORE THESE FIELDS IN TEMPLATE: PATIENT-EXAM
```

To use this template you:

Specify the patient's name to edit.
Select one of the RADIOLOGY EXAM file's entries that point back to that patient.
Edit data within that selected entry in the RADIOLOGY EXAM file.
Return to edit another field in the PATIENT file.
A sample editing session using this INPUT template looks like this:

```
INPUT TO WHAT FILE: PATIENT

EDIT WHICH FIELD: ALL// [PATIENT-EXAM

Select PATIENT NAME: FMPATIENT,11
NAME:  FMPATIENT,11// <Enter>
Select RADIOLOGY EXAM: ?
CHOOSE FROM:
  1.    DEC 4, 1984
  2.    OCT 1, 1985
CHOOSE 1-2: 2
  DATE OF EXAM:  OCT 1, 1985// <Enter>
  RESULTS: NORMAL
ATTENDING PHYSICIAN: FMPATIENT// <Enter>
```

As indicated by this example, the only RADIOLOGY EXAM file entries you were allowed to choose were the two that pointed back to the selected patient (FMPATIENT,11).

Each file, for the purpose of this editing sequence, is considered a subfile of the original, so that when no more fields within the second file are specified, the dialogue falls back to the original file. Having navigated over to a second file, you can use another extended pointer to move to still a third file.

You *cannot* cross file boundaries on input unless you have WRITE access to the file to which you move. This restriction applies to the individual who created this Patient-Exam INPUT template.

Computed field with Multiple Pointer return type

The following example shows the definition of a computed field using a backwards pointer. The result is a computed field with a multiple pointer result type.

```
Select OPTION: MODIFY FILE ATTRIBUTES
Do you want to use the screen-mode version? YES// NO

Modify what File: ZZD TEST FILE1        (6 entries)

Select FIELD: TESTERS
  Are you adding 'TESTERS' as a new FIELD (the 13TH)? No// Y
   FIELD NUMBER: 12//

DATA TYPE OF TESTERS: COMPUTED
'COMPUTED-FIELD' EXPRESSION: NEW PERSON:NAME
By 'NEW PERSON', do you mean the NEW PERSON File, pointing via
its 'ZZD TEST FILE POINTER' field
    ("ZZD" Cross-reference)? Yes// <Enter>  (Yes)
TRANSLATES TO THE FOLLOWING CODE:
S Y(999991,12,80)=$G(D0) X ^DD(999991,12,9.4) S X="" S
D0=Y(999991,12,80)

FIELD IS 'MULTIPLE-VALUED'!

TYPE OF RESULT: MULTIPLE POINTER// <Enter>
POINT TO WHAT FILE: NEW PERSON// <Enter>
....
```

Example: Defining a Computed Multiple Pointer Field

Once the field is defined, it can be used to retrieve the NAME field from the New Person file of entries that point to an entry in the ZZD Test File1 file. Following is an example of its use in the Inquiry to File Entries option.

```
Select OPTION: INQUIRE TO FILE ENTRIES

Output from what File: NEW PERSON// ZZD TEST FILE1   (6 entries)
Select ZZD TEST FILE1 NAME:    SECOND ENTRY
Another one: <Enter>
Standard Captioned Output? Yes// NO  (No)
First Print FIELD: NAME
Then Print FIELD: TESTERS
Then Print FIELD:
Heading (S/C): ZZD TEST FILE1 List// <Enter>
DEVICE: HOME// <Enter>  TELNET

ZZD TEST FILE1 List                  MAR 25,2013@12:57   PAGE 1
NAME                            TESTERS
----------------------------------------------------------------

SECOND ENTRY                    USER,SEVENTEEN
                                USER,THIRTY
                                USER,THIRTYONE
```

Example: Use of Computed Multiple Pointer field

In this case, three Testers are associated with the selected entry.

Chapter 6: Computed Expressions

You can use computed expressions in several places within Fileman to obtain, manipulate, modify, and format data. Computed expressions consist of one or more elements linked together with operators. Most computed expressions return a value after performing the actions you have requested. The way this result is used or displayed depends on where you have used the computed expression.

Syntax

Elements of Computed Expressions

You can use any of the following elements in constructing a computed expression:

- A **field name within the current file** (e.g., RELIGION). The field name can be partially spelled (e.g., REL), if the partial spelling is unambiguous.
- A **field number**, preceded with # (e.g., #3).
- A **literal number**. When used as part of a computed expression, do *not* use quotes (e.g., AGE AT ONSET+20). However, you *must* use quotes if the number will stand alone as a constant (e.g., "3.14159265").
- A **literal text string**, in quotes (e.g., "HELLO").
- A validly formatted **date**, such as 20 JULY 1969, which is punctuated only by spaces.

NOTE: Dashes in a computed expression are interpreted as minus signs. For example, 7-20-1969 would indicate subtraction and be evaluated as -1982.

The word **NUMBER** (or the name of the file followed by the word

NUMBER, such as, PATIENT NUMBER). NUMBER will return the internal entry number of the entry in the file or subfile in question.

The **name of a file followed by the name of a field in that file** (e.g., PATIENT NAME). Like PATIENT NUMBER, this syntax is helpful when it is unclear to which file or subfile an expression is referring. However, this syntax cannot obtain data from **another** file; NAME and PATIENT NAME will return the same data. To obtain data from another file, the extended pointer syntax *must* be used.

A **Fileman function**—e.g., [TODAY or MONTH(DATE OF BIRTH)].

REF: Functions are discussed in the "Fileman Functions" section.

An **extended pointer reference** to fields in another file.

REF: Extended pointers and relational jumping are described in the "Relational Navigation" section.

Operators in Computed Expressions

Computed expressions can consist of a single element. However, often several elements are joined together using operators. Operators are characters that perform some action on elements.

Unary Operators

The simplest operators are the unary operators. They force a numeric interpretation of the element that follows. They can also affect the sign of the resulting number. The unary operators are:

Operator	Description
+	Positive numeric interpretation (sign unchanged)
-	Negative numeric interpretation (sign changed)

Binary Operators

Another set of operators takes two elements, manipulates them, and returns a result. These are called binary operators. You can use the following binary operators in computed expressions:

Operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
\	Integer (truncated) division (e.g., 13\2 = 6)
_	Concatenation (e.g., "AB"_"CDE" = ABCDE)

Boolean Operators

A third set of operators makes a comparison between two elements and returns a true or false value. These are known as Boolean operators. If the outcome of a Boolean operation is true, the value one (1) is returned; if false, zero (0) is returned. You can use these Boolean operators in computed expressions:

Operator	Description
>	Greater than
<	Less than
=	Equal to
]	Follows (in alphabetical order)
[	Contains (e.g., "AB"["A" is true; "A"["AB" is false)
!	Or, either element is true [e.g., (2=3)!(5<10) is true]
&	And, both elements are true [e.g., (2=3)&(5<10) is false]

An apostrophe (') means negation or NOT. It can precede any of the Boolean operators. Thus, **6'>8** is read six is not greater than eight, which is true (a one is returned).

Parentheses in Expressions

In the absence of parentheses, the expression is evaluated strictly left to right. One operator is not given precedence over another. Use parentheses to control the order in which the operations of a computed expression are performed. Expressions within parentheses are evaluated first. Thus, **3+4/2** is 3.5, whereas **3+(4/2)** is 5.

You can also use parentheses to ensure that the enclosed material is treated as an expression when there might be some ambiguity. For example, suppose you want to force a numeric interpretation of the SSN field. You need to use the + unary operator. However, the following will not yield the desired result:

```
SORT BY: +SSN
```

Is the + the unary operator or the sort specifier (meaning that you want to subtotal results by SSN)? In this case, it will be interpreted as the sort specifier. However, if you put the expression in parentheses, the + will definitely be interpreted as an operator:

```
SORT BY: (+SSN)
```

Example of Compound Expression

The following is an example of a computed expression containing several elements and operators:

```
"Beds occupied: "_(NUMBER OF BEDS*OCCUPANCY PERCENTAGE/100)
```

First, the part within the parentheses is evaluated. NUMBER OF BEDS and OCCUPANCY PERCENTAGE are field names. Their contents are multiplied and the result is divided by 100. That result is concatenated with the literal string "Beds occupied: " giving a result like:

```
Beds occupied: 484
```

Data Types in Computed Expressions

When you are working with file data in computed expressions, you *must* consider the appropriateness of the DATA TYPE field value for the operation or function you are using. Here are some notes regarding data types and computed expressions:

SET OF CODES, POINTER TO A FILE, and VARIABLE-POINTER Data Types

These data types are manipulated using the external representations, *not* the internal ones. (The internal value can be accessed using the INTERNAL function.)

DATE/TIME Data Type

The DATA TYPE field value of DATE/TIME usually yields results based on the internal value of the field when used in a computed expression. For example, the computed expression "DATE OF BIRTH: "_DOB, where DOB is a field with a DATA TYPE field value of DATE/TIME, yields "DATE OF BIRTH: 2910713", where 2910713 is the internal representation of the date.

Often, you do not want the internal representation of the date to be used for output. There are alternatives. Continuing with concatenation as an example, you can concatenate a caption with the output of a function (e.g., "DATE OF BIRTH: "_NUMDATE(DOB) yields "DATE OF BIRTH: 07/13/91"). When using the Print File Entries option, you can separately identify the caption like this:

```
    FIRST PRINT FIELD: "DATE OF BIRTH: "
    THEN PRINT FIELD: DOB;X
```

Since DOB was not entered as part of a computed expression, it will produce output in Fileman's external date format: "DATE OF BIRTH: JUL 13, 1991".

You can perform certain arithmetic operations with DATA TYPE field values of DATE/TIME that directly yield useful results:

If you subtract a DATA TYPE field value of DATE/TIME from another DATE/TIME-valued field, the result is the number of days the two differ.

If you add a number to or subtract a number from a DATE/TIME-valued field, the result is a new date. For example, if the DOB field has the value JUL 20, 1969, then the value of the computed expression DOB+30 is AUG 19, 1969.

WORD-PROCESSING Data Type

DATA TYPE fields with a value of WORD-PROCESSING can be manipulated only with the contains ("**[**") operator (e.g., a valid computed expression within the DIAGNOSIS Multiple of our sample PATIENT file (#200) is: HISTORY["poverty"). This Boolean expression is true, if the DIAGNOSIS in question has HISTORY text that contains the string "poverty".

Also, you *cannot concatenate* WORD-PROCESSING-type fields with other values using the concatenation ("_") operator.

Using Functions as Elements in Computed Expressions

You can use recognized functions as an element in any COMPUTED field expression. A function performs an operation that returns a value. These functions are available to all users. Functions can also be added by making entries in the FUNCTION file (#.5). If you examine this file, you will know all of the functions available to you.

REF: For a description on how to add functions, see the "Fileman Functions (Creating)" chapter in the *Fileman Programmer Manual*.

Some functions require an argument or arguments; others are "argumentless." The arguments of the function can be any element, including field name, field number (preceded with the #), quoted literal, or even other functions. The SQUAREROOT function, for example, would take an argument of 64 and return 8. Thus, if the AGE field of a patient has the value 64, the expression SQUAREROOT(AGE) would equal 8.

REF: For information on the syntax and description of the functions exported with Fileman, see the "Fileman Functions" section.

Where to Use

Using Computed Expressions in COMPUTED Fields

One important place where you can use a computed expression is in a field that is computed. The DATA TYPE field value of COMPUTED allows a computed expression to be stored in the data dictionary.

To define a field as COMPUTED, use the Modify File Attributes option and reply to the "DATA TYPE:" prompt with "COMPUTED."

```
Select OPTION: MODIFY FILE ATTRIBUTES
DO YOU WANT TO USER THE SCREEN-MODE VERSION? Yes// NO

MODIFY WHAT FILE: PATIENT

Select FIELD: AGE
  Are you adding 'AGE' as a new FIELD (the 13TH)? Y
   FIELD NUMBER: 13// <Enter>

DATA TYPE OF AGE: COMPUTED
```

You now enter the computed expression that will be stored in the AGE field. In this case, a function (TODAY), a field name (DATE OF BIRTH), and a numeric literal are combined with two arithmetic binary operators to give a numeric result.

```
'COMPUTED-FIELD' EXPRESSION: TODAY-DATE OF BIRTH\365.25
```

TRANSLATES TO THE FOLLOWING CODE:

```
S Y(16033,13,1)=$S($D(^DIZ(16033,D0,0)):^(0),1:""),X=DT S
X=X,X1=X,X2=$P(Y(16033,13,1),U,3),X="" D:X2 ^%DTC:X1 S
X=X\365.25
```

NOTE: You will only see the generated code if you have programmer access.

When creating a COMPUTED field you are given the opportunity to specify the type of result of the computed expression. Note the list of choices available. The the following dialogue is presented:

```
TYPE OF RESULT: STRING// ?

Enter a code from the list.

     Select one of the following:

          S         STRING
          N         NUMERIC
          B         BOOLEAN
          D         DATE
          m         MULTIPLE
          p         POINTER
          mp        MULTIPLE POINTER

TYPE OF RESULT: STRING// NUMERIC
```

For numeric results, you have other questions to answer. First, you are asked to how many decimal places to round the result. In this case, you want the result rounded to a whole number.

```
    NUMBER OF FRACTIONAL DIGITS TO OUTPUT (ONLY ANSWER IF
NUMBER-VALUED): 0
```

Then, you must indicate when you want the rounding to occur.

```
  SHOULD VALUE ALWAYS BE INTERNALLY ROUNDED TO 0 DECIMAL PLACES?
  No// <Enter>  (No)
```

Since the value of a COMPUTED field can be used in other calculations, you need to indicate when rounding should occur. If you accept the default (i.e., "No"), rounding will *not* be done when the COMPUTED field is used in other calculations. A YES answer to this prompt means that you do want the rounded value used in calculations. Usually, you do not want values rounded at interim steps in a series of calculations. Thus, usually, you will accept the "No" default.

When a COMPUTED field is printed, the value will always be rounded to the number of decimal places you specify.

```
  WHEN TOTALLING THIS FIELD, SHOULD THE SUM BE COMPUTED FROM THE
  SUMS OF THE COMPONENT FIELDS? No// <Enter>
```

If your computed expression involves division or multiplication, you will be asked how the field should be totaled. (You can total the values of a field in the Print File Entries option.) A NO answer to this prompt means that the COMPUTED field's expression will be evaluated for each entry and those results will be added. A YES answer means that values of each of the fields in the COMPUTED field's expression will be added first and then the COMPUTED field's expression will be applied to those totals.

For example, suppose **A** and **B** are the names of two fields and **A/B** is a computed expression. The table below shows the results of printing **A**, **B**, and **A/B** with different answers to the "WHEN TOTALLING THIS FIELD, ..." question:

	A	B	A/B (YES: Total from totals of component fields)	A/B (NO: Total from results for each entry)
	10	5	2	2
	100	50	2	2
	2	1	2	2
Total	112	56	[112/56=] 2	[2+2+2=] 6

To summarize, if you want your total to be the ratio or product of the total of the component fields, then answer this question YES. Otherwise, a NO answer is appropriate.

NOTE: The answer to this prompt only affects the Total produced by the Print File Entries option.

When defining a COMPUTED field, you will also be asked:

```
    LENGTH OF FIELD:  8// <Enter>
```

Here you can enter the maximum number of character positions that the field should occupy in output. The default value is eight, even if the COMPUTED field involves FREE TEXT-type fields. Be sure to allocate enough space to accommodate the results. If the COMPUTED field's value is numeric, the entire result is displayed regardless of the requested length.

Another type of result of a Computed Field is Multiple Pointer. A Computed Multiple Pointer occurs when your computed expression relies on a backwards relational jump to another file to retrieve data from that file. Because more than one entry in the pointing file might point to your file, the computed expression can return more than one piece of data. Hence, its result type: “multiple pointer.”

Following is an example of the creation of a Computed Multiple Pointer. In this example, the home file is pointed to by a field in the New Person file called ZZD TEST FILE POINTER. That pointing field must be cross-referenced for the backwards relational navigation to be possible. The NAME field is retrieved form the New Person file.

```
Select OPTION: **MODIFY FILE ATTRIBUTES**
Do you want to use the screen-mode version? YES// **NO**

Modify what File: **ZZD TEST FILE1**    (6 entries)

Select FIELD: **TESTERS**
 Are you adding 'TESTERS' as a new FIELD (the 13TH)? No// **Y**
  FIELD NUMBER: 12// **<Enter>**

DATA TYPE OF TESTERS: **COMPUTED**
'COMPUTED-FIELD' EXPRESSION: **NEW PERSON:NAME**
By 'NEW PERSON', do you mean the NEW PERSON File, pointing via
its 'ZZD TEST FILE POINTER' field
  ("ZZD" Cross-reference)? Yes// **<Enter>**  (Yes)
TRANSLATES TO THE FOLLOWING CODE:
S Y(999991,12,80)=$G(D0) X ^DD(999991,12,9.4) S X="" S
D0=Y(999991,12,80)

FIELD IS 'MULTIPLE-VALUED'!

TYPE OF RESULT: MULTIPLE POINTER// **<Enter>**
POINT TO WHAT FILE: NEW PERSON// **<Enter>**
....
```

You can also use the screen-mode version of Modify File Attributes to define Computed Multiple Pointers.

Now, you can use the TESTERS field in Print Templates, ScreenMan form definitions, or any other place where multiple lines of results are appropriate.

The COMPUTED-type field can be a very useful tool. Having set up such a field, you can then search or sort by it, and also include it in the definition of other COMPUTED-type fields. In the latter case, independence is preserved. Thus, for example, if you define COMPUTED Field #2 in terms of COMPUTED Field #1 and then decide to redefine Field #1, Field #2 will automatically use the new Field #1 calculation. If you try to delete a field that is referenced by a COMPUTED-type field, you will be warned.

Where to Use Computed Expressions "On the Fly"

"On the Fly" Computed Expressions

In addition to permanently storing a computed expression in a data dictionary, there are several places within Fileman's dialogue where you can use a computed expression "**on the fly**":

"PRINT FIELD:" Prompt
"SEARCH FOR FIELD:" Prompt
"SORT BY:" Prompt
Field Value Stuffing
OUTPUT Transforms
Word Processing Windows (| |)

"PRINT FIELD:" Prompt

Whenever, you are within the Print or Search File Entries options, you are asked:

```
FIRST PRINT FIELD:
```

OR

```
THEN PRINT FIELD:
```

You can answer with a computed expression. For example:

```
    FIRST PRINT FIELD: SEX_" "_RELIGION;"";L33
```

This computed expression will return the contents of the SEX and RELIGION fields separated by a space. Notice that you can follow the computed expression with print qualifiers: **;""** to suppress the column heading and **;L33** to indicate that the COMPUTED field length can be 33 characters long.

A user with programmer access can also enter M code at this prompt. The M code *must* have a WRITE statement for anything that is to be written to the report.

"SEARCH FOR FIELD:" Prompt

In the Search File Entries option, you can answer the

```
SEARCH FOR FIELD:
```

prompt with a computed expression.

If the expression is Boolean (i.e., its value is either true or false), you will not be asked the condition of the search, because the computed expression itself specifies that condition.

A user with programmer access can also enter M code at this prompt. The M code *must* set the variable X to whatever is to be compared against the search value.

"SORT BY:" Prompt

In Print or Search File Entries options, you can answer the

```
SORT BY:
```

prompt with a computed expression. If you use a Boolean computed expression, you will not be asked for the "START WITH" parameters.

A user with programmer access can also enter M code at this prompt. The M code *must* set the variable X to the sort value.

Thus, if you want to print a list of the names of all patients who are Baptists, you could enter:

```
Select OPTION: PRINT FILE ENTRIES

OUTPUT FROM WHAT FILE: PATIENT
SORT BY: RELIGION="BAPTIST"
   WITHIN RELIGION="BAPTIST", SORT BY: <Enter>
FIRST PRINT FIELD: NAME
```

This is a common way to select certain records for printing.

Field Value Stuffing

In the Enter or Edit File Entries option, you can follow the // or /// specifiers with computed expressions. The expression is evaluated for the entry you are inputting and used as a variable stuff value.

Suppose you want to put the current contents of a patient's NEXT OF KIN field into the BENEFICIARY field, with a notation that this value is UNVERIFIED, for all patients who do not have a value in the BENEFICIARY field. The dialogue would look like this:

```
Select OPTION: ENTER OR EDIT FILE ENTRIES
INPUT TO WHAT FILE: PATIENT
EDIT WHICH FIELD: BENEFICIARY///NEXT OF KIN_" (UNVERIFIED)"
THEN EDIT FIELD: <Enter>

Select PATIENT NAME: ^LOOP
    EDIT ENTRIES BY: BENEFICIARY=""
 WITHIN BENEFICIARY="", EDIT ENTRIES BY: <Enter>
```

This example uses two "on the fly" expressions: one in answer to the "EDIT ENTRIES BY:" prompt (which is essentially a SORT BY for looping) and one as the forced default value for the BENEFICIARY input field.

BENEFICIARY="" is a Boolean (true-false) computed expression that means "The BENEFICIARY value equals null."

After the previous dialogue, the names of such patients would be printed out, and their BENEFICIARY value would automatically be set equal to their NEXT OF KIN field value, concatenated with a space followed by "(UNVERIFIED)."

OUTPUT Transforms

OUTPUT transforms change the way a field is displayed when printed. Frequently, the OUTPUT transform will contain a computed expression that alters the data stored internally in the field. A simple OUTPUT transform that converts the internally stored date into MM/DD/YY format is:

```
    DATE OF BIRTH OUTPUT TRANSFORM: NUMDATE(DATE OF BIRTH)
```

If an OUTPUT transform is applied to a field, the result of the transform will be used if that field is used in another computed expression. For example, if DATE OF BIRTH is used in a PRINT template, the "transformed" value will be output:

```
    THEN PRINT FIELD: NAME_"'S BIRTHDAY: "_DATE OF BIRTH
```

The result of this computed expression would be similar to:

```
    ONE FMPATIENT'S BIRTHDAY: 03/07/42
```

Word Processing Windows (| |)

When entering text into a DATA TYPE field with a value of WORD-PROCESSING, you can insert a computed expression within a |**Window**|. This expression will be evaluated at the time the WORD-PROCESSING-type field is printed. If the expression is meaningful, its value will replace the |**Window**| in the printed output.

For example, you could embed within the text of the HISTORY WORD-PROCESSING-type field a |**Window**| containing a COMPUTED field expression:

```
    HISTORY:
    1> PATIENT IS A |SEX_" "_RELIGION| WHO HAS NO
    2> APPARENT PROBLEMS.
```

When this field is printed for a patient who has a SEX value of MALE and a RELIGION value of CATHOLIC, the output would look like:

```
    PATIENT IS A MALE CATHOLIC WHO HAS NO
    APPARENT PROBLEMS.
```

Chapter 7: Fileman Functions

How to Use Fileman Functions

This chapter lists each Fileman Function, including syntax and simple examples of their use. You can use them in any computed expression.

REF: For more information on computed expressions, see the "Computed Expressions" section.

A function performs an operation that returns a value. Many functions are included with Fileman; you can also add functions by making entries in the FUNCTION file (#.5).

REF: For a description on how to add functions, see the "Fileman Functions (Creating)" section in the *Fileman Programmer Manual*.

Some functions require an argument or arguments; others are "argumentless." The arguments of the function can be any element, including field name, field number (preceded with the #), quoted literal, or even other functions. The SQUAREROOT function, for example, would take an argument of 64 and return 8. Thus, if the AGE field of a patient has the value 64, the expression SQUAREROOT(AGE) would return 8.

NOTE: If there is an output transform on a field, the function code is applied to the field after it has been transformed. In most cases, if a field has an output transform, you should therefore use the syntax FUNCTION_NAME(INTERNAL(FIELD_NAME)), rather than FUNCTION_NAME(FIELD_NAME).

Documentation Conventions for Fileman Functions

While studying this chapter's functions, syntax, and examples, you will encounter the following conventions:

Convention	Description
"	In the format arguments: Indicates mandatory quotation marks. **NOTE:** If you enter a literal string as an argument, quotation marks are also necessary.
=>	In examples: Indicates the output of the function.
[ ]	In examples: Indicates information about the outcome of the function.
boldface type	Indicates specific reference to an argument.

FUNCTION(argument, . . .) is the general format. You *must* enter the function's name in uppercase; the case of the arguments depends on the circumstances. Arguments are always surrounded by parentheses.

Fileman Functions

REF: For a description of each function, see the "Fileman Functions" chapter in the **HTML version** of the *Fileman Advanced Programmer Manual* located on the Fileman Home Page at: http://VISTA.med.va.gov/fileman/index.asp

Category	Function
Date/Time	BETWEEN DATE DAYOFWEEK MID MINUTES MONTH MONTHNAME NOON NOW NUMDATE NUMDATE4 NUMDAY NUMMONTH NUMYEAR NUMYEAR4 RANGEDATE TIME TODAY YEAR
Environmental	BREAKABLE CLOSE SITENUMBER USER
File and File Data	COUNT DUPLICATED FILE INTERNAL LAST MAXIMUM MINIMUM nTH NEXT PREVIOUS TOTAL

Category	Function
Mathematical	ABS BETWEEN MAX MIN MODULO SQUAREROOT
Printing Related Functions	IOM PAGE
String	DUP LOWERCASE PADRIGHT REPLACE REVERSE STRIPBLANKS TRANSLATE UPPERCASE
Temporary Data Storage	PARAM and SETPARAM VAR and SET
M-Related Functions	$A[SCII] $C[HAR] $E[XTRACT] $F[IND] $H[OROLOG] $I[O] $J[OB] $J[USTIFY] $L[ENGTH] $P[IECE] $R[ANDOM] $S[ELECT] $S[TORAGE] $X $Y

Date/Time Functions

BETWEEN

Format:	BETWEEN(d1,d2,d3)
Parameters:	d1, d2, and d3 are dates or date expressions: d1 is the date being tested. d2 is one limit for the test. d3 is the other limit for the test.
Use:	This Boolean function determines if d1 is within the limits defined by d2 and d3. If d1 is within this range, a value of 1 (true) is returned; otherwise, 0 (false) is returned. If d1 equals d2 or d3, 1 (true) is returned.
Examples:	Select OPTION: **SEARCH FILE ENTRIES** OUTPUT FROM WHAT FILE: BUILD// **<Enter>** -A- SEARCH FOR BUILD FIELD: **BETWEEN(DATE DISTRIBUTED,1JAN2000,1JAN2001)** -B- SEARCH FOR BUILD FIELD: IF: A// **<Enter>** BETWEEN(DATE DISTRIBUTED,1JAN2000,1JAN2001)

DATE

Format:	DATE(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This date function returns the date portion of a date/time expression.
Example:	DATE(NOW) => AUG 21,1991

REF: For tips on displaying date-valued elements such as this function in computed expressions (e.g., printing), see the "Data Types in Computed Expressions" topic in the "Computed Expressions" section.

DAYOFWEEK

Format:	DAYOFWEEK(datexp)
Parameters:	datexp is an expression with date/time value.
Use:	This function returns the day of the week of the date in datexp.
Example:	DAYOFWEEK(DATE OF BIRTH) => TUESDAY

MID

Format:	MID
Parameters:	(none)
Use:	This argumentless function returns the current date with a 24:00 time stamp. It represents tonight at midnight.
Example:	MID => AUG 23,1991 24:00

REF: For tips on displaying date-valued elements such as this function in computed expressions (e.g., printing), see the "Data Types in Computed Expressions" topic in the "Computed Expressions" section.

MINUTES

Format:	MINUTES(datexp1,datexp2)
Parameters:	datexp1 and datexp2 are date/time expressions. Time stamps are not necessary.
Use:	This function returns the number of minutes that datexp1 is after datexp2. If no time is associated with a date/time expression, DATE@12:00 A.M. is used.
Examples:	MINUTES(MID,NOW) => 832 MINUTES(MID,TODAY) => 1440

MONTH

Format:	MONTH(datexp)
Parameters:	datexp is a date/time expression.
Use:	This function returns the month and year from a date/time valued expression.
Example:	MONTH(DATE OF BIRTH) => AUG 1943

MONTHNAME

Format:	MONTHNAME(n)
Parameters:	n is an expression that evaluates to an integer from 1 through 12.
Use:	This function returns the full name of the month corresponding to n.
Examples:	MONTHNAME(4) => APRIL MONTHNAME(+$E(DATE OF BIRTH,4,5)) => APRIL [Function $E extracts the 4th and 5th digits from a date stored in FileMan internal format: YYYMMDD.]

NOON

Format:	NOON
Parameters:	(none)
Use:	This argumentless function returns today's date with a time stamp of 12:00.
Example:	NOON => AUG 23,1991 12:00

REF: For tips on displaying date-valued elements such as this function in computed expressions (e.g., printing), see the "Data Types in Computed Expressions" topic in the "Computed Expressions" section.

NOW

Format:	NOW
Parameters:	(none)
Use:	This argumentless function returns the current date and time.
Example:	NOW => AUG 23,1991 11:23

REF: For tips on displaying date-valued elements such as this function in computed expressions (e.g., printing), see the "Data Types in Computed Expressions" topic in the "Computed Expressions" section.

NUMDATE

Format:	NUMDATE(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the date in datexp in MM/DD/YY format.
Example:	NUMDATE(DATE OF BIRTH) => 03/07/49

NUMDATE4

Format:	NUMDATE4(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the date in datexp in MM/DD/YYYY format.
Example:	NUMDATE4(DATE OF BIRTH) => 03/07/1949

NUMDAY

Format:	NUMDAY(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the day of the month in datexp as a number.
Example:	NUMDAY(DATE OF BIRTH) => 7 [DATE OF BIRTH = March 7, 1949]

NUMMONTH

Format:	NUMMONTH(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the month in datexp as a number.
Example:	NUMMONTH(DATE OF BIRTH) => 3 [DATE OF BIRTH = March 7, 1949]

NUMYEAR

Format:	NUMYEAR(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the last two digits of the year in datexp as a number.
Example:	NUMYEAR(DATE OF BIRTH) => 49 [DATE OF BIRTH = March 7, 1949]

NUMYEAR4

Format:	NUMYEAR4(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the four digit year in datexp as a number.

Example:	NUMYEAR4(DATE OF BIRTH) => 1949 [DATE OF BIRTH = March 7, 1949]

RANGEDATE

Format:	RANGEDATE(datexp1,datexp2,datexp3,datexp4)
Parameters:	datexp1 is a date valued expression beginning the first range of dates. datexp2 is a date valued expression ending the first range of dates. datexp3 is a date valued expression beginning the second range of dates. datexp4 is a date valued expression ending the second range of dates.
Use:	This function returns the number of days that the two ranges of dates overlap.
Example:	RANGEDATE(DATE OF BIRTH,NOW,20 JUL 1981,20 JUL 1982) => 366

TIME

Format:	TIME(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns time from datexp in 12 hour format with AM/PM.
Example:	TIME(NOW) => 1:15 PM

TODAY

Format:	TODAY
Parameters:	(none)

Use:	This argumentless function returns today's date.
Example:	TODAY => AUG 26,1991

REF: For tips on displaying date-valued elements such as this function in computed expressions (e.g., printing), see the "Data Types in Computed Expressions" topic in the "Computed Expressions" section.

YEAR

Format:	YEAR(datexp)
Parameters:	datexp is an expression with a date/time value.
Use:	This function returns the year from datexp.
Example:	YEAR(DATE OF BIRTH) => 1949

Environmental Functions

BREAKABLE

Format:	BREAKABLE(n)
Parameters:	n is a number or numeric expression with a value of 1 or 0.
Use:	This function returns nothing. When used within a PRINT template, this function determines whether or not <Ctrl-C> can be used to break out of a report print. If n = 1, <Ctrl-C> will break out; if n = 0, it will not. Under default conditions, <Ctrl-C> will break you out. The value of n is returned.
Example:	BREAKABLE(0) =>0; [<Ctrl-C> is disabled]

CLOSE

Format:	CLOSE(device)
Parameters:	device is an open device, in the form of a valid argument for an M Close command.
Use:	This function should only be used within Fileman code when Kernel is unavailable. It closes the specified device.

SITENUMBER

Format:	SITENUMBER
Parameters:	(none)
Use:	This argumentless function returns your site's identifying number that was entered during Fileman initialization and stored in ^DD("SITE",1). (Do *not* use this function to retrieve a VA Institution Station Number.)
Example:	SITENUMBER => 99

USER

Format:	USER("attribute")
Parameters:	attribute is one of these codes: # user's DUZ value (the user's number) N user's name I user's initials T user's title NN user's nickname **NOTE:** These codes *must* be surrounded by quotes within the function.
Use:	This function returns information about the currently logged on user. The information comes from the NEW PERSON file (#200). **NOTE:** This function will not work if you are using Fileman without a NEW PERSON file in ^VA(200,.

Example:	USER("#") => 160

File and File Data Functions

COUNT

Format:	COUNT(fname)
	COUNT(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple in your current file. In the second format: fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	This function counts the number of entries in a file or in a Multiple. You can count the lines in a word processing field by using the first format with the word processing field name as the fname. If the second format is used, the number of entries with non-null values in field is returned.
Examples:	COUNT(PATIENT) -> 1349 [the number of entries in the PATIENT file] COUNT(PATIENT:PROVIDER) => 1288 [number of patients with providers recorded]

DUPLICATED

Format:	DUPLICATED(field)
Parameters:	field is the name of a field (or a field number preceded by #). The field *must* be a cross-referenced field.
Use:	This function, when used on any cross-referenced field, will find all duplicates within a given file or determine whether a specific entry is duplicated.

	Returns one of the possible Boolean values: 1=field value is duplicated in another entry. ""=field value is unique.

Examples:	Example using the Search File Entries option to perform a search on the example file named ZZINDIVIDUAL:
	`Select OPTION: SEARCH FILE ENTRIES`
	`OUTPUT FROM WHAT FILE: ZZINDIVIDUAL// <Enter>`
	`  -A- SEARCH FOR ZZINDIVIDUAL FIELD: DUPLICATED(NAME)`
	`  -B- SEARCH FOR ZZINDIVIDUAL FIELD:`
	`IF: A// <Enter>  DUPLICATED(NAME)`
	`STORE RESULTS OF SEARCH IN TEMPLATE: <Enter>`
	`SORT BY: NAME// <Enter>`
	`START WITH NAME: FIRST// <Enter>`
	`FIRST PRINT FIELD: NUMBER`
	`THEN PRINT FIELD: NAME`
	`THEN PRINT FIELD: <Enter>`
	`Heading (S/C): ZZINDIVIDUAL SEARCH// <Enter>`
	`DEVICE: <Enter>  Telnet Terminal    Right Margin: 80// <Enter>`
	`ZZINDIVIDUAL SEARC    MAR 18,2008  14:44  PAGE 1`
	`NUMBER        NAME`
	`------------------------------------------------`
	`5             FMPATIENT,ONE`
	`15            FMPATIENT,ONE`
	`                         2 MATCHES FOUND.`
	Another example for using DUPLICATED, this time using Option PRINT FILE ENTRIES, would be if you wanted to print the name with three asterisks in front of it if it were a duplicated name:
	`  FIRST PRINT FIELD: $S(DUPLICATED(NAME):"***",1:"")_NAME`

FILE

Format:	FILE(vpointer)
Parameters:	vpointer is the label or field number of a variable pointer field.
Use:	This function returns the name of the file to which a variable pointer points for a particular entry.
Example:	FILE(PROVIDER) => STAFF PROVIDERS

INTERNAL

Format:	INTERNAL(field)
Parameters:	field is the label of a field or a field number preceded by #.
Use:	This function returns the internally stored value of the field for a particular entry. It is useful in obtaining the internally stored (instead of displayed) DATA TYPE field value of any of the following: POINTER TO A FILE VARIABLE-POINTER DATE/TIME SET OF CODES
Examples:	INTERNAL(PROVIDER) => 136;VA(200, INTERNAL(SEX) => m

LAST

Format:	LAST(fname)
	LAST(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple-valued field in your current file. In the second format:

	fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	This function returns the last entry in a file or in a Multiple identified by fname. If the second format is used, the last entry with a non-null value in field is returned. The last entry is the one with the highest internal entry number; the function does not analyze the values of the entries.
Examples:	LAST(DIAGNOSIS) => Sepsis [last entry in this Multiple field] LAST(DIAGNOSIS:OCCURRENCES) => 3

MAXIMUM

Format:	MAXIMUM(fname)
	MAXIMUM(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple in your current file. In the second format: fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	With the first format, this function returns the largest value from the .01 field of the file or Multiple identified by fname. The second format returns the largest value from field. The function works only if the internally stored values of the entries are numeric. Thus, you can use numeric or date valued fields. Also, free text fields will work if the stored values are numbers. Computed fields with numeric results can be used. Pointer fields will return the value from the pointed-to file.
Examples:	MAXIMUM(APPOINTMENT) => FEB 25,1991

	[APPOINTMENT is a Multiple-valued DATE/TIME field] MAXIMUM(PATIENT:AGE) => 93 [AGE is a field in the current file, PATIENT]

MINIMUM

Format:	MINIMUM(fname)
	MINIMUM(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple-valued field in your current file. In the second format: fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	This function returns the smallest value from the file's .01 field or from the Multiple identified by fname. The second format returns the smallest value from field. (See MAXIMUM for limits of use.)
Examples:	MINIMUM(APPOINTMENT) => MAR 1,1979 [APPOINTMENT is a Multiple-valued DATE/TIME field] MINIMUM(PATIENT:AGE) => 18

nTH

Format:	The syntax of this function is different because the function's name is defined by the user. The name is a number followed by an ordinal number suffix.
	nTH(fname)
	nTH(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple in your current file.

	In the second format: fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	This function returns the nth entry in a file or in a Multiple identified by fname. If the second format is used, the value of the specified field associated with the nth entry in fname is returned. The nth entry is determined by the internal entry number; the function does not analyze the values of the entries. When used with the second format, the nth subentry with a non-null value is returned.
Examples:	2ND(DIAGNOSIS) => Angina Pectoris [the second entry in the DIAGNOSIS Multiple] 10TH(ADMISSION:ADMISSION DATE) => JAN 2,1990 [ADMISSION DATE associated with the tenth ADMISSION]

NEXT

Format:	NEXT(field)
Parameters:	field is a field's number preceded by a # or a field's label from the current file or Multiple.
Use:	This function returns the value for the field identified by field in the next entry. The next entry is determined by internal entry number. No analysis of the value of entries is done. If there are no more entries, the function returns null.
Example:	NEXT(AGE AT ONSET) => 56 [the value of AGE AT ONSET for the next entry in the Subfile]

PREVIOUS

Format:	PREVIOUS(field)
Parameters:	field is a field's number preceded by a # or a field's label from the current file or Multiple.
Use:	This function returns the value for the field identified by field in the previous entry. The previous entry is determined by internal entry number. No analysis of the value of entries is done. If there is no prior entry, the function returns null.
Example:	PREVIOUS(AGE AT ONSET) => 29 [the value of AGE AT ONSET for the prior entry in the Subfile]

TOTAL

Format:	TOTAL(fname)
	TOTAL(fname:field)

Parameters:	In the first format, fname is the name of a file or of a Multiple-valued field in your current file. In the second format: fname is the name of your current file or Multiple. field is the name of a field (or a field number preceded by #) in fname.
Use:	With the first format, this function totals the values of the .01 field of a Multiple or file identified by fname. The second format totals the values in field. The field being totaled *must* have numeric values.
Example:	"$"_TOTAL(VISIT COST) => $569.32 [VISIT COST is a Multiple]

Mathematical Functions

ABS

Format:	ABS(n)
Parameters:	n is a number or an expression with a numeric value.
Use:	This mathematical function will return the value of n without a sign; it gives the absolute value of n.
Example:	ABS(-23.87) => 23.87

BETWEEN

Format:	BETWEEN(n1,n2,n3)
Parameters:	n1, n2, and n3 are numbers or numeric expressions: n1 is the number being tested. n2 is one limit for the test. n3 is the other limit for the test.
Use:	This Boolean function determines if n1 is within the limits defined by n2 and n3. If n1 is within this range, a value of 1 (true) is returned; otherwise, 0 (false) is returned. If n1 equals n2 or n3, 1 (true) is returned.
Examples:	BETWEEN(OCCURRENCES,5,10) => 0 [OCCURRENCES is a field with value = 3] BETWEEN(-3,-10,0) => 1

MAX

Format:	MAX(n1,n2)
Parameters:	n1 and n2 are numbers or numeric expressions.
Use:	This function returns the larger of n1 and n2. Date/time field values can be used resulting in the most recent date/time being returned.
Examples:	MAX(54,23) => 54 MAX(DATE OF BIRTH,TODAY) => AUG 23,1991

MIN

Format:	MIN(n1,n2)
Parameters:	n1 and n2 are numbers or numeric expressions.
Use:	This function returns the smaller of n1 and n2. Date/time field values can be used resulting in the earliest date/time being returned.
Examples:	MIN(54,23) =>23 MIN(DATE OF BIRTH,TODAY) => NOV 1,1938

MODULO

Format:	Format: MODULO(n1,n2)
Parameters:	n1, n2 are numbers or numeric expressions: n1 is the dividend. n2 is the divisor.
Use:	This mathematical function returns the remainder when n2 is divided into n1; it performs modulo division.
Example:	MODULO(54,5) => 4

SQUAREROOT

Format:	SQUAREROOT(n)
Parameters:	n is a numeric expression greater than 0.
Use:	This mathematical function returns the square root of n.
Example:	SQUAREROOT(9) => 3

Printing Related Functions

IOM

Format:	IOM
Parameters:	(none)
Use:	This argumentless function returns the number of columns for the present output device.
Example:	IOM/2 => 0

PAGE

Format:	PAGE
Parameters:	(none)
Use:	This argumentless function returns the current page number when output is being printed.
Example:	"Page "_PAGE => Page 23 [the 23rd page of output]

String Functions

DUP

Format:	DUP(string,n)
Parameters:	string is any string of characters or an expression yielding a string of characters. n is a positive integer or a numeric expression.
Use:	This function returns a string of characters n characters long. If string is less than n characters long, the characters in string will be repeated until the output string is n characters in length.
Example:	DUP(DIAGNOSIS,3) => Ang [value of DIAGNOSIS = Angina Pectoris] DUP("_",IOM) => ______________________ [line drawn will have a length equal to the value of IOM]

LOWERCASE

Format:	LOWERCASE(string)
Parameters:	string is an expression yielding alphabetic characters.
Use:	This function will change uppercase characters in string to lowercase except for the first character and the first character after a punctuation mark. A space is a punctuation mark; thus, the first letter of a word will not be changed. String cannot be a word-processing field; the contents of a word-processing field will be unaffected.
Example:	LOWERCASE("FMPATIENT,20") => Fmpatient,20

PADRIGHT

Format:	PADRIGHT(string,n)
Parameters:	string is a string or string expression to be printed. n is the total size of the output string.
Use:	This function will pad string on the right with spaces to make a string n characters long. If string is longer than n characters, the entire string will be returned; this function will not truncate.
Examples:	PADRIGHT("Peter",10) => Peter [five spaces after the 'r'] PADRIGHT(CITY,15) => San Juan Capistrano

REPLACE

Format:	REPLACE(string,oldstring,newstring)
Parameters:	string is the string expression that will be changed. oldstring is a string expression containing the character(s) in string that will be replaced. newstring is a string expression containing the character(s) that will replace those in oldstring.

Use:	This function returns the input string with all occurrences of the oldstring changed to the newstring. The oldstring and newstring can be any length. They do not have to be equal in length.
Examples:	REPLACE("abracadabra","ab","*") => *racad*ra REPLACE("Name is: XXX","XXX",NAME) => Name is: FMPATIENT,21

REVERSE

Format:	REVERSE(string)
Parameters:	string is a string expression.
Use:	This function returns the characters in string in reverse order.
Example:	REVERSE(NAME) => neB,nilknarF

STRIPBLANKS

Format:	STRIPBLANKS(string)
Parameters:	string is a string expression.
Use:	This function removes leading and trailing spaces from string.
Example:	STRIPBLANKS(" Waste no space ") => Waste no space [no leading or trailing spaces]

TRANSLATE

Format:	TRANSLATE(string,"oldchar","newchar")
Parameters:	string is a string expression to be changed. oldchar are the character(s) to be translated. newchar are the character(s) to replace the oldchar.

Use:	This function alters string by changing each character in oldchar into the character in the corresponding position in newchar. The translation is one character for one character.
Examples:	TRANSLATE("08261991","123","ABC") => 08B6A99A TRANSLATE(NAME,"F","f") => fMPATIENT,fORTY-ONE

UPPERCASE

Format:	UPPERCASE(string)
Parameters:	string is an expression with alphabetic characters.
Use:	This function will change lowercase characters in string to uppercase. String cannot be a word processing field; the contents of a word processing field will be unaffected. If the user’s language is not English and if that language has code in the Language File (#.85) to change lowercase characters to uppercase, this function will change the characters according to the language-specific instructions.
Example:	UPPERCASE("VISTA") => VISTA

Temporary Data Storage Functions

PARAM

Format:	PARAM("parameter")
Parameters:	parameter has been assigned a value by the SETPARAM function.
Use:	This function works with the SETPARAM function. It returns the value that has been given to parameter by use

	of the SETPARAM function.
Example:	PARAM("AGE") => 45

SETPARAM

Format:	SETPARAM(value,"parameter")
Parameters:	value is an expression to be evaluated. parameter is a string 1 to 30 characters long identifying a storage location to hold value.
Use:	This function works with the PARAM function. It returns nothing. Value is stored in parameter for later reference.
Example:	SETPARAM(TODAY-DATE OF BIRTH\365,"AGE") => [no output; result of the expression put into "AGE"]

VAR

Format:	VAR("variable")
Parameters:	variable is a variable in the local symbol table.
Use:	This function returns the value of variable. The variable can be one that you set using the SET function.
Examples:	VAR("COUNT") => 1 [1 is the current value of COUNT] VAR("DUZ") => 160

SET

Format:	SET(value,"variable")
Parameters:	value is an expression to be evaluated. variable is a local variable name used to hold the value of value.
Use:	This function returns value's value. In addition, the value

	is placed in a local variable. Variable should be namespaced to avoid conflict with other local variables. You can use this function only if you have programmer's access.
Example:	SET(1,"COUNT") => 1 [this would put 1 into the variable COUNT]

M-Related Functions

$A[**SCII**]

Format:	$A(string,n)
Parameters:	string is a string of characters or an expression yielding a string. n is an integer or expression yielding an integer.
Use:	The function returns the numeric ASCII value of the character in position n within string. If n is not specified, the value of the first character is returned.
Examples:	$A(NAME,4) => 77 [NAME is SHAM,SAM THE] $A("Get the value") => 71

$C[HAR]

Format:	$C(n, . . .)
Parameters:	n is an integer or an expression yielding an integer.
Use:	This function returns the character corresponding to the ASCII value of n. If more than one n is specified in the argument, a string of characters will be returned.
Examples:	$C(100) => d $C(99,100,101) => cde

$E[XTRACT]

Format:	$E(string,n1,n2) $E(string,n) $E(string)
Parameters:	string is a string expression. n, n1, and n2 are positive integers or expressions yielding positive integers.
Use:	This function returns a substring from string. If you use only string as a argument, the first character is returned. If you specify one n, the character in that position in the string is returned. If you specify n1 and n2, a string starting at n1 and ending at n2 is returned.
Examples:	$E(NAME,3,7) => ith,A [NAME is FMPATIENT,21] $E(NAME,2) => m $E(NAME) => S

$F[IND]

Format:	$F(string,target) $F(string,target,n)
Parameters:	string is a string expression. target is the character(s) or an expression yielding the character(s) to be searched for. n is a positive integer or an expression yielding a positive integer.
Use:	This function returns the character position in string following the target. If n is specified as a third argument, the search for target is begun after character position n. If target is not found, 0 is returned.
Examples:	$F("FMPATIENT,21",",") => 7 $F(NAME,",",7) => 0 [NAME has value of FMPATIENT,21]

$H[OROLOG]

Format:	$H
Parameters:	(none)
Use:	This system variable returns the date and time in internal M format. The format is number of days since December 31, 1840, followed by a comma followed by the number of seconds since midnight.
Example:	$H => 55032,48780

$I[O]

Format:	$I
Parameters:	(none)
Use:	This system variable returns the current device. It may return the operating system's designation of the current device.
Example:	$I => _LTA9239

$J[OB]

Format:	$J
Parameters:	(none)
Use:	This system variable returns your current job number.
Example:	$J => 666172581

$J[USTIFY]

Format:	$J(string,n) $J(n1,n2,n3)
Parameters:	In the first format, string is a string expression; n is an integer representing width of field.

	In the second format, n1 is a numeric expression; n2 is an integer representing the width of field; n3 is the number of decimal places to output with the number.
Use:	In the first format, the function returns string right justified within a field that has a width of n. If string is longer than n, there is no truncation. In the second format, the function returns n1 right justified in a field that has a width of n2. There will be n3 decimal places to the right of the decimal point.
Example:	$J(NAME,20) => FMPATIENT,21 [12 spaces preceding the 'S'] "$"_$J(PRESCRIPTION COST,8,2) => $ 25.88 [3 spaces preceding the '2']

$L[ENGTH]

Format:	$L(string) $L(string,delimiter)
Parameters:	string is a string expression. delimiter is a character (or characters) or an expression yielding a character (or characters) that divides the string into pieces.
Use:	In the first format, the function returns the number of characters in string. In the second format, the function returns the number of pieces into which delimiter divides the string. If delimiter does not exist within string, 1 is returned.
Examples:	$L(PROVIDER) => 11 [PROVIDER is FMPROVIDER,5] $L(PROVIDER,",") => 2 [same PROVIDER]

$P[IECE]

Format:	$P(string,"delimiter",n) $P(string,"delimiter",n1,n2) $P(string,"delimiter")
Parameters:	string is a string expression. delimiter is a character (or characters) or an expression yielding a character (or characters) that divides the string into pieces. n, n1, and n2 are positive integers or expressions evaluating to positive integers.
Use:	The function returns a part of string. String is divided into substrings by delimiter. In the first format, the nth substring is returned. In the second format, the substrings starting with n1 and ending with n2 are returned. The delimiters between those substrings are also returned. In the third format, the first substring (i.e., the one preceding the first occurrence of delimiter) is returned.
Examples:	$P("FMPATIENT,22",",",2) => 22 $P(PHONE,"-",2,3) => 943-2109 $P(PHONE,"-") => 510

$R[ANDOM]

Format:	$R(n)
Parameters:	n is a positive integer or an expression evaluating to a positive integer.
Use:	This function returns a randomly generated integer from the range of 0 through n-1.
Example:	$R(5000) => 1076

$S[ELECT]

Format:	$S(test:value,test:value,...)
Parameters:	expression is an expression that can be evaluated as True or False (not zero or zero). value is any expression that can yield a value.
Use:	Each value is associated with the test from which it is separated by a colon. The function returns the evaluation of the value associated with the first test that evaluates as true (i.e., not equal to zero). Any number of test:value pairs can be used; however, one of the tests *must* evaluate as true. To assure that one test will always evaluate as true, the last test is usually the literal 1.
Examples:	$S("SIX FMPROVIDER, Ph.D."["M.D.":"He is a medical doctor.",1:"He is not a medical doctor.") => He is not a medical doctor. $S(OCCURRENCES>3:"Chronic Condition",OCCUR-RENCES>0: "Non Chronic Condition",1:"No Occurrences Recorded") => Chronic Condition [Here the contents of the OCCURRENCES field is being tested. If the first test (>3) is true (as in this example), the result of the second test (>0) is not relevant.]

$S[TORAGE]

Format:	$S
Parameters:	(none)
Use:	This system variable returns the number of bytes of free space available for use. Its meaning varies with the M implementation.
Example:	$S => 52672

$X

Format:	$X
Parameters:	(none)
Use:	This system variable returns the current X coordinate (column) location of the cursor or print head. If the application that moved the cursor did not update the value of $X, the value of $X will not be reliable.
Example:	$X => 43

$Y

Format:	$Y
Parameters:	(none)
Use:	This system variable returns the current Y coordinate (row) location of the cursor. Like $X, its reliability depends on the controlling application.
Example:	$Y => 6

Let’s Make Some Files

Chapter 8: Creating Files and Fields

Creating a File

To create a new file, use the Modify File Attributes option and enter the new file name (from 3 to 45 characters in length) when asked:

```
MODIFY WHAT FILE:
```

And respond with YES when asked "Are you adding 'xxxxxxxx' as a new FILE?" (where "xxxxxxxx" represents the new file name).

Naming a New File

File names should be chosen so that they can easily be distinguished from each other:

```
MODIFY WHAT FILE: ENT CLINIC PATIENT
```

TIP: If different people are creating files, it can be helpful to include their initials, nicknames, or other identifying phrases within the file name.

NOTE: The name for your new file should not contain any arithmetic or string operators like **+ - * / \ _** or punctuation like a colon ("**:**").

An internal number *must* be assigned to this new file. You are prompted with the next available internal file number. You can either simply press the **Enter** key (a null response) to accept that number, or enter a number *not* already assigned to a file.

Your new file is now initialized. Fileman creates the NAME field (field number .01), which is free text, 3 to 30 characters in length, non-numeric, and has no leading punctuation. You will see the following message:

```
A FreeText NAME Field (#.01) has been created.
```

The .01 field's definition can be modified like any other field: it does not have to be FREE TEXT; its label need not be NAME; and so forth. The .01 field should be the key attribute of an entry used to identify it and to order the entries for lookups. (However, entries can be identified and ordered by other fields through cross-references.)

After creating a new file, you can define any number of fields for the new file, as described in the "Creating Fields" topic below.

REF: For an example of a new file using the Modify File Attributes option, see the "Examples of File and Field Creation" section.

Creating Fields

For any file, you can create fields describing logically related data that pertains to entries in that file. When created, every file automatically receives one field: a NAME field (the #.01 field). You *must* explicitly define any other fields. All such definitions are made (and changed) with the Modify File Attributes option.

When you create a file, after you give the new file a name, you are asked:

```
Select FIELD:
```

Enter a new field name and respond with YES when Fileman asks:

```
Are you adding 'xxxxxxxx' as a new FIELD?
```

The "xxxxxxxx" represents the name of the field you entered. After answering YES to this prompt, you are now ready to specify what sort of data the new field will contain. A field can only be *one* type of data; the choices are listed in the following section.

Field Data Types

There are nine data types:

No.	Data Type	Description
1	DATE/TIME	Dates with or without time stamps.
2	NUMERIC	DATA TYPE fields of NUMERIC, including dollar values.
3	SET OF CODES	Codes that represent values (e.g., 1=MALE/2=FEMALE).
4	FREE TEXT	A single alphanumeric string of characters.
5	WORD-PROCESSING	A Multiline document of text.
6	COMPUTED	A virtual field, values not stored.
7	POINTER TO A FILE	Referencing an entry in some other file.
8	VARIABLE-POINTER	Referencing an entry in a defined set of files.
9	MUMPS	Used by programmers to enter M code.

You are asked for the DATA TYPE field value for any field you are creating. You *must* pick one of these nine choices. Data validation checks are then asked depending on the DATA TYPE field value entered. For some data types, a default "HELP" prompt is automatically composed.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For examples of entering a field attributes in Screen Mode, see the "Examples of File and Field Creation" section.

Screen Mode Field Editing

When using the Modify File Attributes option you can make your field entries in the traditional Scrolling Mode or choose to create, modify, or review a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

To make your entries in Screen Mode, simply press the **Enter** key to accept the default response at the "Do you want to use the screen-mode version? YES//" prompt.

The following example illustrates using Screen Mode when editing the .01 field of the ORDER file (#100):

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER

Select FIELD: .01 <Enter>  ORDER #
```

You will then be taken into Screen Mode where you can edit the properties of the field, as shown below:

```
                           Field #.01 in File #100
FIELD LABEL: ORDER #                                 DATA TYPE...
NUMERIC

          TITLE:
          AUDIT:
AUDIT CONDITION:
    READ ACCESS:
  DELETE ACCESS:
   WRITE ACCESS:
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... NO

      MANDATORY: YES
    HELP-PROMPT: Enter the order number.
XECUTABLE HELP:
```

```
________________________________________________________________________________
NOTE THAT THIS FIELD'S DEFINITION IS NOT EDITABLE


                                          Press <F1>H for help    Insert
```

Using the Screen Mode version of this option, after entering the field name or number (e.g., .01 field) after the "Select FIELD:" prompt, you are presented with a Screenman form (screen) that can be reviewed and edited like any other. In this example, the most important field on this screen is the DATA TYPE field in the upper right corner; it is a required entry. In this example, required entries in Screen Mode are indicated by a caption with a different color and an underline.

NOTE: Screen Mode highlights the captions for required fields with an underline. However, depending on your terminal or terminal emulator software and your personal preferences, the form of the highlight can vary (e.g., some emulators will highlight required field captions in reverse video, a different color, with an underline, or any combination of highlights).

REF: For more information on Screenman forms and Screen Mode, see the "Screenman" chapter in the *Fileman User Manual*.

In this case, the DATA TYPE field has been defined as NUMERIC and is *not editable* as indicated by the message displayed near the bottom of the screen (i.e., "NOTE THAT THIS FIELD'S DEFINITION IS NOT EDITABLE"); this is because a programmer has previously edited the definition in a special way. However, unlike the DATA TYPE field, the value of the HELP-PROMPT field (i.e., "Enter the Order number."), which is the message that is displayed to users when they enter a single question mark ("**?**") while editing the ORDER # field, can be edited. The DESCRIPTION and TECHNICAL DESCRIPTION fields are multi-line WORD-PROCESSING fields; to edit them, press the **Enter** key and a separate screen opens (i.e., a "popup" window). The DESCRIPTION is displayed to users who enter two question marks ("**??**") while editing the ORDER # field. The TECHNICAL DESCRIPTION, however, is for internal documentation only.

DATE/TIME Data Type

A DATA TYPE field defined as DATE/TIME allows you to enter a minimum and maximum date. You can also indicate whether the date can be entered with an imprecise date (e.g., JUL 1969) or with the time-of-day (e.g., JUL 20@4). Fileman does *not* accept dates *before* 1700.

For example, when defining a DATA TYPE field value of DATE/TIME, you are asked the following questions:

```
Select FIELD: DATE OF BIRTH
  Are you adding 'DATE OF BIRTH' as a new FIELD? NO// Y

DATA TYPE OF DATE OF BIRTH: DATE/TIME
EARLIEST DATE (OPTIONAL): 1/1/1860
LATEST DATE: 1963
CAN DATE BE IMPRECISE (Y/N):  YES// <Enter>
CAN TIME OF DAY BE ENTERED (Y/N):  NO// <Enter>
```

If you reply YES to the "CAN TIME OF DAY BE ENTERED (Y/N): NO//" prompt, you would then be asked "CAN SECONDS BE ENTERED?".

```
WILL DATE OF BIRTH FIELD BE MULTIPLE:  No// <Enter>
IS DATE OF BIRTH ENTRY MANDATORY (Y/N):  NO// Y
...
'HELP'-PROMPT:  TYPE A DATE BETWEEN 1/1/1860 AND 1963
           Replace <Enter>
DESCRIPTION:
  1> <Enter>
```

A default help prompt is automatically written for you with the DATA TYPE field of DATE/TIME. You can change this prompt using the "Replace ... With" syntax.

NOTE: This help information is displayed when the user inputs a single question mark ("**?**") when editing this field.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value as DATE/TIME in Screen Mode, see the "Examples of File and Field Creation" section.

NUMERIC Data Type

A DATA TYPE field defined as NUMERIC requires you to enter the lowest and highest values allowed, the maximum number of decimal digits allowed, and to state whether dollar values are allowed (e.g., $33).

For example, when defining a DATA TYPE field value as NUMERIC, you are asked the following questions:

```
Select FIELD: AGE AT ONSET
  Are you adding 'AGE AT ONSET' as a new FIELD? No// Y

DATA TYPE OF AGE AT ONSET: NUMERIC
INCLUSIVE LOWER BOUND: 0
INCLUSIVE UPPER BOUND: 100
IS THIS A DOLLAR AMOUNT (Y/N):  NO// <Enter>
```

If you answer YES at the "IS THIS A DOLLAR AMOUNT (Y/N):" prompt, Fileman would allow users to precede input data with a dollar sign ("**$**") and show up to two decimal places.

```
MAXIMUM NUMBER OF FRACTIONAL DIGITS: 0// <Enter>
WILL AGE AT ONSET FIELD BE MULTIPLE? No// <Enter>
IS AGE AT ONSET ENTRY MANDATORY (Y/N): NO// <Enter>
....
'HELP'-PROMPT:  Type a Number between 0 and 100, 0 Decimal
Digits
           Replace <Enter>
DESCRIPTION:
  1> <Enter>
```

A default help prompt is automatically written for you with the DATA TYPE field value of NUMERIC. You can change this prompt using the "Replace ... With" syntax.

NOTE: This help information is displayed when the user inputs a single question mark ("**?**") when editing this field.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of NUMERIC in Screen Mode, see the "Examples of File and Field Creation" section.

SET OF CODES Data Type

A DATA TYPE field defined as a SET OF CODES can be used to restrict a user to just a few possible values (e.g., YES or NO). When defining a DATA TYPE field value of SET OF CODES, enter a valid code and a translation of what each code means. The user can enter the code, the full meaning, or a portion of the full meaning. If the field is set up to require only a one-character response, this data type can simplify the user's data entry.

Fileman has only a limited amount of space to store the codes and their external values. If the limit is exceeded, you are told "TOO MUCH!!--SHOULD BE A 'POINTER', NOT 'SET'." The DATA TYPE field value of SET OF CODES is sometimes referred to as a SET.

For example, when defining a DATA TYPE field value as SET OF CODES, you are asked the following questions:

```
Select FIELD: SEX
  Are you adding 'SEX' as a new FIELD? No// Y <Enter>

DATA TYPE OF SEX: SET OF CODES
INTERNALLY-STORED CODE: m <Enter>  WILL STAND FOR: MALE
INTERNALLY-STORED CODE: f <Enter>  WILL STAND FOR: FEMALE
INTERNALLY-STORED CODE: <Enter>
WILL SEX FIELD BE MULTIPLE: No// <Enter>
IS SEX ENTRY MANDATORY (Y/N):  No// Y
...
'HELP'-PROMPT: <Enter>
DESCRIPTION:
  1> <Enter>
```

In this example, "**m**" stands for Male and "**f**" stands for Female. Numbers as well as alphabetic characters can be used.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of SET OF CODES in Screen Mode, see the "Examples of File and Field Creation" section.

FREE TEXT Data Type

A DATA TYPE field defined as FREE TEXT allows you to enter the maximum and minimum allowable string length of the FREE TEXT data. You can also enter an M PATTERN MATCH that input data will have to match.

For example, when defining a DATA TYPE field value as FREE TEXT, you are asked the following questions:

```
Select FIELD: DIAGNOSIS
  Are you adding 'DIAGNOSIS' as a new FIELD? No// Y

DATA TYPE OF DIAGNOSIS: FREE TEXT
MINIMUM LENGTH: 3
MAXIMUM LENGTH: 30
(OPTIONAL) PATTERN MATCH (IN 'X'): <Enter>
```

The PATTERN MATCH is written in M code. If input data violates the PATTERN MATCH or the Minimum/Maximum lengths, the data is not accepted and the user is shown the help prompt information.

```
WILL DIAGNOSIS FIELD BE MULTIPLE? No// Y
IS DIAGNOSIS ENTRY MANDATORY(Y/N): NO// <Enter>
SHOULD USER SEE AN "ADDING A NEW DIAGNOSIS?" MESSAGE FOR NEW
   ENTRIES (Y/N): N
HAVING ENTERED OR EDITED ONE DIAGNOSIS, SHOULD USER BE ASKED
   ANOTHER (Y/N): Y
```

With these specifications, the user is not given a confirming message when new subentries are added to the Multiple. The user is allowed to enter several diagnoses in a row for a given patient.

```
....
....
'HELP'-PROMPT:  Answer must be 3-30 characters in length.
           Replace <Enter>
DESCRIPTION:
  1> <Enter>
```

A default help prompt is automatically written for you with the DATA TYPE field value of FREE TEXT. You can change this prompt using the "Replace ... With" syntax.

NOTE: This help information is displayed when the user inputs a single question mark ("**?**") when editing this field.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value as FREE TEXT in Screen Mode, see the "Examples of File and Field Creation" section.

WORD-PROCESSING Data Type

A DATA TYPE field defined as WORD-PROCESSING allows entry of unlimited free-text data. The data can be edited, formatted, and printed with word-processing text editors.

REF: For a description of Fileman's native text editors, see the "Screen Editor" and "Line Editor" chapters in the *Fileman User Manual*.

For example, when defining a DATA TYPE field value as WORD-PROCESSING, you are asked the following questions:

```
Select FIELD: HISTORY
  Are you adding 'HISTORY' as a new FIELD? No// Y

DATA TYPE OF HISTORY: WORD-PROCESSING
SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE? Yes// <Enter>
....
....
```

```
'HELP'-PROMPT: SUBJECTIVE NARRATIVE OF PATIENT'S PROBLEM HISTORY
DESCRIPTION:
  1> <Enter>
```

If you answer YES to the "SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE" question, text is automatically wrapped at word boundaries to fit in the column in which it is being printed. Usually, this is the preferred way to print text.

TIP: When it is important that lines of text be printed exactly as they were entered, answer NO to the "SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE" question. Thus, text is output in no-wrap mode. You would probably want a spreadsheet or a restaurant menu printed in no-wrap mode.

NOTE: If the column in which no-wrap text is being printed is too short to accommodate the line of text, your printer may break the line in the middle of words or otherwise destroy the formatting of the text.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of WORD-PROCESSING in Screen Mode, see the "Examples of File and Field Creation" section.

COMPUTED Data Type

When a DATA TYPE field is defined as COMPUTED, its value is determined at the time the field is accessed. This computation is based on an expression stored in the data dictionary. The field value (or data) itself is not stored in the data dictionary. The COMPUTED expression is constructed using field names, literals or constants, functions, and operators.

REF: For a complete explanation of these elements, see the "Computed Expressions" section.

NOTE: The functions referred to above are Fileman functions stored in the FUNCTION file (#.5), not M functions. A developer with programmer access can also enter M code in a COMPUTED field. The M code *must* set the variable "X" to the COMPUTED field value.

For example, when defining a DATA TYPE field value as COMPUTED, you are asked the following questions:

```
Select FIELD: AGE
  Are you adding 'AGE' as a new FIELD? No// Y

DATA TYPE OF AGE: COMPUTED

'COMPUTED-FIELD' EXPRESSION: TODAY-(DATE OF BIRTH)\365.25

....
NUMBER OF FRACTIONAL DIGITS TO OUTPUT (ONLY ANSWER IF NUMBER-
VALUED): 2
```

For this example, we assume the DATE OF BIRTH field was previously created. Thus, we can reference it in the " 'COMPUTED-FIELD' EXPRESSION". Also, the "NUMBER OF FRACTIONAL DIGITS TO OUTPUT" question is asking whether you should enter the number of digits that should normally appear to the right of the decimal point when this field is displayed.

```
SHOULD VALUE ALWAYS BE INTERNALLY ROUNDED TO 2 DECIMAL PLACES?
No// Y
WHEN TOTALLING THIS FIELD, SHOULD THE SUM BE COMPUTED FROM
       THE SUMS OF THE COMPONENT FIELDS? No// <Enter>
LENGTH OF FIELD: 8// <Enter>
```

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of COMPUTED in Screen Mode, see the "Examples of File and Field Creation" section.

POINTER TO A FILE Data Type

A DATA TYPE field defined as a POINTER TO A FILE requires you to enter the name or number of the pointed-to file and that file *must* already exist (defined previously).

For example, when defining a DATA TYPE field value as POINTER TO A FILE, you are asked the following questions:

```
Select FIELD: RELIGION
  Are you adding 'RELIGION' as a new FIELD? No// Y

DATA TYPE OF RELIGION: POINTER TO A FILE
POINT TO WHICH FILE: RELIGION
```

The file that is pointed to *must* already exist on your system. If you enter a single question mark ("**?**") at the "POINT TO WHICH FILE:" prompt, you will be presented with a list of the available files.

```
SHOULD 'ADDING A NEW RELIGION FILE ENTRY' ("LAYGO")
BE ALLOWED WHEN ANSWERING THE 'RELIGION' QUESTION?  No// <Enter>
```

By answering NO to this prompt, users who are editing patient data are not able to add a new entry on the fly to the RELIGION file. This prompt depends on whether you have LAYGO access to the file or not.

```
WILL RELIGION FIELD BE MULTIPLE? No// <Enter>
IS RELIGION ENTRY MANDATORY (Y/N): NO// <Enter>
....
'HELP'-PROMPT: <Enter>
DESCRIPTION:
  1> <Enter>
```

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of POINTER TO A FILE in Screen Mode, see the "Examples of File and Field Creation" section.

VARIABLE-POINTER Data Type

A DATA TYPE field defined as a VARIABLE-POINTER (as with the POINTER TO A FILE DATA TYPE) requires you to enter the names or numbers of the pointed-to files and those files *must* already exist (defined previously). Additionally, an order, message, and prefix *must* be associated with each file.

For example, when defining a DATA TYPE field value as VARIABLE-POINTER, you are asked the following questions:

```
Select FIELD: PROVIDER
  Are you adding 'PROVIDER' as a new FIELD? No// Y

DATA TYPE OF PROVIDER: VARIABLE-POINTER
Select VARIABLE POINTER: PROVIDER
```

You answer the "Select VARIABLE POINTER:" prompt with the name or number of an existing file.

```
  Are you adding 'PROVIDER' as a new VARIABLE-POINTER? No// Y

VARIABLE-POINTER:  PROVIDER// <Enter>
MESSAGE: Staff Provider
ORDER: 1
PREFIX: S
SHOULD USER BE ALLOWED TO ADD A NEW ENTRY: NO
```

The MESSAGE is part of the online help associated with the VARIABLE-POINTER field when a single question mark ("**?**") is entered during editing. In this example, the PROVIDER file MESSAGE is associated with "Staff Provider."

The several pointed-to files are searched based on their ORDER. In this example, since the ORDER for the PROVIDER VARIABLE-POINTER is one, the PROVIDER file is the first file searched.

The PREFIX field is used to reference a particular pointed-to file. To see the entries in a particular file, you would enter that file's PREFIX followed by a period and a question mark at the VARIABLE-POINTER's field name (e.g., in this example, entering "**S.?**" would give you the option to list the entries in the PROVIDER file). If you want to refer to only one of the several pointed-to files, put that file's PREFIX followed by a period at the VARIABLE-POINTER's field name (e.g., in this example, entering "**S.**" would refer you to the PROVIDER file).

By answering NO to the "SHOULD USER BE ALLOWED TO ADD A NEW ENTRY:" prompt, the user is *not* allowed to add new entries on the fly to the PROVIDER file (the same as the RELIGION field entered as a POINTER TO A FILE). If you had answered YES, then new entries could be added to the PROVIDER file.

```
Select VARIABLE-POINTER: 16 <Enter>  PERSON
  Are you adding 'PERSON' as a new VARIABLE-POINTER? (the 2ND)?
No// Y
VARIABLE-POINTER:    PERSON// <Enter>
MESSAGE: Other Provider
ORDER: 2
PREFIX: O
SHOULD USER BE ALLOWED TO ADD A NEW ENTRY: YES
```

In this example, a second VARIABLE-POINTER was created to the PERSON file (#16). By answering YES to the "SHOULD USER BE ALLOWED TO ADD A NEW ENTRY:" prompt, users who are editing patient data are allowed to add entries to the PERSON file.

```
Select VARIABLE-POINTER: <Enter>
```

You stop identifying files for a VARIABLE-POINTER by simply pressing the **Enter** key without any additional entries at the "Select VARIABLE-POINTER:" prompt.

```
WILL PROVIDER FIELD BE MULTIPLE?  No// <Enter>
IS PROVIDER ENTRY MANDATORY (Y/N):  NO// <Enter>
'HELP'-PROMPT: <Enter>
DESCRIPTION:
  1> <Enter>
```

After entering both VARIABLE-POINTERS, when you enter a single question mark ("**?**") at the "PROVIDER" field prompt, you will see the following help message:

```
PROVIDER: ?
     Enter one of the following:
       S.EntryName to select a Staff Provider
       O.EntryName to select a Other Provider

     To see the entries in any particular file type <Prefix.?>
```

In this example, if you simply enter a name at the "PROVIDER:" prompt, then the system will search each of the VARIABLE-POINTER field files for the name you have entered.

If a match is found, the system will ask you if it is the correct entry. However, if you know the file the entry should be in, then you can speed processing by using the following syntax to select an entry:

```
PREFIX.entry name
MESSAGE.entry name
File Name.entry name
```

NOTE: You do *not* need to enter the entire file name or message to direct the lookup. Using the first few characters will suffice.

NOTE: You can review and change a file's field attributes easily by running the Modify File Attributes option in Screen Mode.

REF: For an example of entering a DATA TYPE field value of VARIABLE-POINTER in Screen Mode, see the "Examples of File and Field Creation" section.

MUMPS Data Type

Those with programmer access can define a field with a DATA TYPE field value of MUMPS. This MUMPS-valued field is designed specifically to contain executable M code. The code entered into this kind of field is verified to be valid M code that conforms to VA programming standards.

MUMPS-valued fields are usually used in files that are part of developer tools systems. For example, the OPTION file (#19) is part of the MENU MANAGEMENT system used in the VA for assigning menus and associated actions to each computer user. The ENTRY ACTION field on the OPTION file (#19) is defined as a MUMPS-type field. This field allows a developer who is creating an option to enter M code to do any setup and initialization that is needed before the end user can do the action allowed by the option.

Multiple-Valued Field (Multiples)

When you create a DATA TYPE field value of any of the following:

DATE/TIME
NUMERIC
SET OF CODES
FREE TEXT
POINTER TO A FILE
VARIABLE-POINTER

After entering type-specific information, you are asked the following:

```
WILL FIELD BE MULTIPLE:  NO//
```

Answering YES to this prompt, means that:

- There can be more than one occurrence of a data value for this field in the entry (e.g., more than one DIAGNOSIS in a PATIENT file entry)
- Subfields can later be associated with this field (e.g., each DIAGNOSIS could have a DATE OF ONSET)
- The "MULTIPLE:" prompt is not asked for DATA TYPE field values of

WORD-PROCESSING, since, by definition, such types take Multiline values.

Two special questions are asked about Multiple-valued fields:

- SHOULD USER SEE AN 'ADDING NEW ENTRY?' MESSAGE FOR NEW ENTRIES (Y/N) Answering NO here means that the new Diagnosis (or whatever the Multiple is named) gets added without a verification prompt being asked.
- HAVING ENTERED OR EDITED ONE, SHOULD USER BE ASKED ANOTHER (Y/N): Answering YES means that the user is prompted to put in several diagnoses, one right after the other. NO means that the user is not prompted to enter a second value.

Making a Field Mandatory

For all DATA TYPE field values except COMPUTED, as you create the field, you are asked the following:

```
IS "xxxxxxxx" ENTRY MANDATORY (Y/N):  NO//
```

The "*xxxxxxxx*" represents the name of the field. If you answer YES, the user of your file will *not* be allowed to skip the field without entering data for a particular entry.

Field Number Sequences

It is often useful to sequence the fields so that when you are using the Enter or Edit File Entries option and you ask to edit ALL fields, you will see the field questions presented in a natural order. If you want to add a CURRENT AGE field to our PATIENT file and place it between DATE OF BIRTH (#2) and RELIGION (#3), the dialogue would be:

```
Select FIELD: 2.5
  Are you adding a new FIELD:  No// Y
```

```
      LABEL: CURRENT AGE
      FIELD NUMBER:  2.5// <Enter>
```

You could have specified any number between two and three.

NUMBER (.001) Field

All files have unique numbers associated with each of their entries. Defining the NUMBER field allows you to use the Internal Entry Number (IEN, also called the record number) as you would any other field. Usually, this means that someone (like Herr Doktor Ludwig Koechel in the case of the MOZART WORK file) has gone to the trouble of creating a numbering scheme for the entries. If you wish to set up a file in which a unique Internal Entry Number is always matched with each entry Name, you can do so by creating a field numbered .001 for the file:

```
Select FILE: MOZART WORK
Select FIELD: .001
  Are you adding a new FIELD? No// YES
LABEL: KOECHEL NUMBER
FIELD NUMBER: .001// <Enter>

DATA TYPE OF KOECHEL NUMBER: NUMERIC
INCLUSIVE LOWER BOUND: 1
INCLUSIVE UPPER BOUND: 626
IS THIS A DOLLAR AMOUNT (Y/N):  NO// <Enter>
MAXIMUM NUMBER OF FRACTIONAL DIGITS:  0// <Enter>
HELP PROMPT: Type a Number between 1 and 626, 0 Decimal Digits.
         Replace <Enter>
DESCRIPTION:
  1> <Enter>
```

The previous dialogue is what would normally create a NUMERIC-valued field. In this case, we are describing the file's PRIMARY KEY, or Internal Entry Number.

Once such a .001 field is defined, you can create a new file entry that might look like this:

```
Select MOZART WORK: EINE KLEINE NACHTMUSIK
  Are you adding a new MOZART WORK? No// YES
KOECHEL NUMBER: 525
```

More importantly, with a .001 field defined, an entry in the file can always be looked up by the Internal Entry Number (IEN), irrespective of any other cross-referencing that exists for the file. Thus:

```
Select MOZART WORK: 525 <Enter>  EINE KLEINE NACHTMUSIK
```

Record Numbers *must* always be positive and canonic—that is, they *cannot* contain alpha suffixes, leading zeros, or trailing fractional zeros.

Forced Lookups Using Numbers

Number-meaningful lookups can be forced by prefixing the numeric input with the ` (accent grave). If 55 FMPATIENT's Internal Entry Number in the PATIENT file is 355, he could be identified as follows:

```
Select PATIENT NAME: `355 <Enter>  FMPATIENT,5
```

Incidentally, the .001 field example (above) illustrates how using the Modify File Attributes option can force a field to have a particular number (.001 in this case). It is done just by entering the new Number first, and then the new Label.

Changing and Deleting Fields

Changing Field Attributes

After creating a field in a file, you can return to **change** or **delete** the field within the Modify File Attributes option, simply by entering the field name (or number) when asked:

```
Select FIELD:
```

When you return to the field in this option, you are able to change a field's:

- Label (Name)
- Title (long form of its name)
- Audit and Audit Conditions (to indicate which fields should be audited)
- Read/Delete/Write
- Source
- Destination
- Group
- Description of the field, a WORD-PROCESSING field (what the user sees after entering two question marks)
- Technical Description

After you are presented with these attributes of the field, you can change the attributes defined during the initial definition of the field as illustrated in the "Creating Fields" topic above.

For example, let's say you have created an SSN field (Social Security Number) in the PATIENT file and would now like to edit the field:

```
Select FIELD: SSN
LABEL:  SSN// <Enter>
TITLE: Social Security Number
AUDIT: YES, ALWAYS
AUDIT CONDITION: <Enter>
```

REF: Auditing is described in the section.

```
READ ACCESS (OPTIONAL): <Enter>
DELETE ACCESS (OPTIONAL): <Enter>
WRITE ACCESS (OPTIONAL): <Enter>
```

REF: Control of various kinds of access to files is described in the section.

```
SOURCE: <Enter>
Select DESTINATION: <Enter>
Select GROUP: **DEMOG**
```

A GROUP is a shorthand way for the user to refer to several fields at once when using the Print File Entries or the Enter or Edit File Entries options. Here, SSN is being assigned to the DEMOG group.

```
DESCRIPTION:
1>An entry is required.  If you do not know this patient's
Social
2>Security Number, enter '000000000' to indicate the number is
3>unknown.
EDIT Option: <Enter>
TECHNICAL DESCRIPTION:
1> <Enter>
```

The DESCRIPTION and TECHNICAL DESCRIPTION attributes document the use and meaning of the field. The information in DESCRIPTION is shown to the user when two question marks are entered at the "EDIT Option:" prompt. When initially creating a field, you are prompted for the DESCRIPTION field after the 'HELP'-PROMPT. The TECHNICAL DESCRIPTION is displayed only when the data dictionary is printed.

NOTE: Versions of Fileman prior to Version 21.0 allowed you to also enter a Help Frame for field documentation; that attribute is no longer supported.

```
DATA TYPE OF SSN:  FREE TEXT// <Enter>
MINIMUM LENGTH: 9// <Enter>
MAXIMUM LENGTH: 9// <Enter>
(OPTIONAL) PATTERN MATCH (IN 'X'): X?9N// <Enter>
IS SSN ENTRY MANDATORY (Y/N): Y// <Enter>
'HELP'-PROMPT:  ANSWER MUST BE 9 CHARACTERS IN LENGTH
  Replace ... With  Enter 9 numbers without dashes, e.g.,
666456789.
  Replace <Enter>
      Enter 9 numbers without dashes, e.g., 666456789.
```

To illustrate the use of GROUPs, add the NAME, DATE OF BIRTH, and SEX fields into the DEMOG group:

```
Select FIELD: NAME
```

```
LABEL:   NAME// ^GROUP
Select GROUP: DEMOG
DESCRIPTION:
1> <Enter>
TECHNICAL DESCRIPTION:
1> <Enter>

DATA TYPE OF NAME:   FREE TEXT// ^
```

DATE OF BIRTH and SEX can be added to the GROUP in the same way. Now, when using the Enter or Edit File Entries option, you could say:

```
EDIT WHICH FIELD: DEMOG
 1    DEMOG    NAME
 2    DEMOG    SEX
 3    DEMOG    DATE OF BIRTH
EDIT WHICH FIELD: <Enter>

Select PATIENT NAME: FMPATIENT,55
NAME:   FMPATIENT,ONE// <Enter>
SEX:   MALE// <Enter>
DATE OF BIRTH:   JAN 3, 1955// <Enter>
```

Changing a Field's DATA TYPE Value

Within the Modify File Attributes option, you can change the DATA TYPE field value itself. There are limitations on the sort of changes you can make. These are listed below.

You *must* be very careful in making such changes if you already have file data entered, because there is no guarantee that the old data will match the newly specified criteria (e.g., field length). However, if you do change a field definition, you are asked if you want existing data checked for inconsistencies. A list of any discrepancies is printed. If more than one discrepancy is found, you can save the list of discrepant entries in a template. To generate this list later, use the Verify Fields option on the Utility Functions menu [DIUTILITY].

The following restrictions apply to changing the definitions of existing DATA TYPE field values in a file:

- Multiple-valued fields *cannot* be changed to single-valued fields or vice versa. Multiple-valued fields can only be defined when creating a field.
- COMPUTED fields *cannot* be changed to other types of fields or vice versa.
- WORD-PROCESSING-type fields should only be changed into Multiple-valued FREE TEXT fields.
- Only a Multiple-valued FREE TEXT field can be changed into a WORD-PROCESSING field, and only if no other subfields are defined for that field.
- POINTER TO A FILE type fields *cannot* be changed to VARIABLE-POINTER type fields or vice versa.

Deleting an Existing Field

Deleting a field and its definition is done by deleting the field Name (LABEL). Delete the field by typing the at-sign ("@") after the display of a field's LABEL when using the Modify File Attributes option:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: PATIENT

Select FIELD: SEX
LABEL:  SEX// @
    SURE YOU WANT TO DELETE THE ENTIRE 'SEX' FIELD? YES
OK TO DELETE 'SEX' FIELDS IN THE EXISTING ENTRIES? YES
```

CAUTION: If you answer NO to the "OK TO DELETE" question, data conflicts may occur in the future, if you create new fields. It is advisable to always delete existing entries. Only a programmer can delete the entries after you have answered NO.

Examples of File and Field Creation

The following examples of creating files/fields or editing fields in a file are illustrated using Screen Mode.

File Creation
DATE/TIME Field
SET OF CODES Field
FREE TEXT Field
WORD-PROCESSING Field
COMPUTED Field
POINTER TO A FILE Field
VARIABLE-POINTER Field
Creating a Multiple
Subfields
NUMERIC Subfield

NOTE: These examples assume that the user does *not* have programmer access.

REF: For explanations of additional capabilities available to the programmer, see the "Advanced File Definition" chapter in the *Fileman Programmer Manual*.

File Creation

The following example illustrates the file definition dialogue you see when creating a new file using the Modify File Attributes option. In this case, we will create the ORDER file (#100, a standard Fileman file):

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER
  Are you adding 'ORDER' as a new FILE?  No// Y
   FILE NUMBER:  99000// 100
```

```
...SORRY, HOLD ON...
     A FreeText NAME Field (#.01) has been created.


Select FIELD: NAME
```

Fileman prompts you to enter the file name. If it is a new file, Fileman will ask you to confirm that you want to add a new file. The default file number to the left of the "//" (e.g., 99000) is related to a site number that is assigned to your computer when Fileman is initialized. This file number is within the range of numbers assigned to your site. Be sure to follow local policies when assigning file numbers.

As you can see from this example, when a file is created a field with the label NAME and number .01 is automatically created. When creating a new file, you can change the definition of this field in the same way you can change the definition of any other field.

When you select a field (e.g., the NAME field), you will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                           Field #.01 in File #100
FIELD LABEL: NAME                                 DATA TYPE... FREE TEXT

          TITLE:
          AUDIT:
AUDIT CONDITION:
    READ ACCESS:
  DELETE ACCESS:
   WRITE ACCESS:
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... NO

     MANDATORY: YES
   HELP-PROMPT: NAME MUST BE 3-35 CHARACTERS, NOT NUMERIC OR
STARTING WITH PU
XECUTABLE HELP:
```

```
COMMAND:                               Press <F1>H for help    Insert
```

DATE/TIME Fields

In the following example, the DATA TYPE field for the RELEASE DATE/TIME field (#.68) in the ORDER file (#100) has a DATA TYPE field value of DATE/TIME:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: RELEASE DATE/TIME
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                           Field #.68 in File #100
FIELD LABEL: RELEASE DATE/TIME                DATA TYPE... DATE

  ┌------------------------------------------------------------┐
  |                EARLIEST DATE: _                            |
AU|                  LATEST DATE:                              |
  |        CAN DATE BE IMPRECISE: NO                           |
  |   CAN TIME OF DAY BE ENTERED: YES                          |
  |       CAN SECONDS BE ENTERED: YES                          |
  |             IS TIME REQUIRED: YES                          |
 D|                                                            |
  └------------------------------------------------------------┘

                       IS THIS FIELD MULTIPLE... NO

     MANDATORY: NO .
   HELP-PROMPT: Enter the Date/time this order was release4d to
the service
XECUTABLE HELP:
________________________________________________________________


COMMAND:                                  Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. DATE/TIME-type fields do not have any required entries. You can accept all the default values within the "popup" window, simply by closing the window by pressing the "F1-C" keys.

NOTE: To delete the entire field, enter an at-sign ("@") at the "FIELD LABEL:" prompt.

SET OF CODES Field

In the following example, the DATA TYPE field for the FLAGGED field (#.61) in the ORDER file (#100) has a value of SET OF CODES:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: FLAGGED
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                           Field #.61 in File #100
FIELD LABEL: FLAGGED                                    DATA TYPE... SET
     ┌----------------------------------------------------------┐
     |CODE: 0      WILL STAND FOR: NO                            |
     |CODE: 1      WILL STAND FOR: YES                           |
AUDIT|CODE:        WILL STAND FOR:                               |
    R|CODE:        WILL STAND FOR:                               |
  DEL|CODE:        WILL STAND FOR:                               |
   WR|CODE:        WILL STAND FOR:                               |
     |CODE:        WILL STAND FOR:                               |
 DESC|CODE:        WILL STAND FOR:                               |
     |CODE:        WILL STAND FOR:                               |
     |CODE:        WILL STAND FOR:                               |
     |CODE:        WILL STAND FOR:                               |
     |CODE:        WILL STAND FOR:                               |
   HE|CODE:        WILL STAND FOR:                               |
cation
XECUT└-----------------------------------------------------------┘
────────────────────────────────────────────────────────────────


COMMAND:                          Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. SET OF CODES-type fields require input of all the allowable "Internal" values (i.e., "CODE" prompt), and their "External" equivalents (i.e., "WILL STAND FOR" prompt).

Using the **<ArrowUp>** and **<ArrowDown>** keys makes it easy to edit the codes. It is permissible to leave a blank row, but every Internal Code (i.e., "CODE," on the left) *must* have a corresponding External Code ("WILL STAND FOR," on the right), and vice versa.

FREE TEXT Field

In the following example, the DATA TYPE field for the REASON FOR FLAG field (#.67) in the ORDER file (#100) has a value of FREE TEXT:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: REASON FOR FLAG
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                        Field #.67 in File #100
FIELD LABEL: REASON FOR FLAG             DATA TYPE... FREE TEXT
        ┌--------------------------------------------------------┐
        │        MINIMUM LENGTH: 1                                │
        │        MAXIMUM LENGTH: 80                               │
AUDIT C│  PATTERN MATCH (IN 'X'): X'?1P.E                         │
    REA└--------------------------------------------------------┘
```

```
  DELETE ACCESS:
   WRITE ACCESS:
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... NO

     MANDATORY: NO
   HELP-PROMPT: Enter the reason for the flag.
XECUTABLE HELP:
________________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. FREE TEXT-type fields require input for the "MINIMUM" and "MAXIMUM" lengths of the field. In this example, users *must* enter from 1 to 80 characters for this field.

If included, the PATTERN MATCH *must* be written in M. For example, you can insure that the data value will not start with a punctuation character by entering a PATTERN MATCH of "**X'?1P.E**". This is a good check to make on a field that is allowed to be just one character in length. Sometimes, users may mistype and answer a field prompt with "/" (same key as "**?**") or some other meaningless punctuation character. A PATTERN MATCH check such as "**X'?1P.E**" will keep that kind of mistake out of your database.

Carets ("^") in a FREE TEXT Field

If you are going to have carets ("^") in a FREE TEXT field, it is advisable to create the field on a node by itself. You should create the field as usual, but when Fileman asks for the ^-PIECE POSITION, reply with E1,<maximum length>, as shown below:

```
Select FIELD: SPECIAL SITUATION
  Are you adding 'SPECIAL SITUATION' as a new FIELD (the 3RD)?
No// Y
   FIELD NUMBER: 11// 50
```

```
DATA TYPE OF SPECIAL SITUATION: FREE TEXT
MINIMUM LENGTH: 3
MAXIMUM LENGTH: 200
(OPTIONAL) PATTERN MATCH (IN 'X'):
WILL MY TRY FIELD BE MULTIPLE? No// <Ent

SUBSCRIPT: 0// 50
^-PIECE POSITION: 1// E1,200
IS MY TRY ENTRY MANDATORY (Y/N): NO// <Enter>
....
'HELP'-PROMPT: Answer must be 3-200 characters in length.
           Replace <Enter>
XECUTABLE 'HELP':
DESCRIPTION:
  No existing text
  Edit? NO//
```

The E1,200 means that the field occupies positions 1 through 200 of the node.

WORD-PROCESSING Field

In the following example, the DATA TYPE field for the ORDER TEXT field (#.11) in the ORDER file (#100) has a value of WORD-PROCESSING:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: ORDER TEXT
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
            Field #.01 in Sub-File #772.02 of File #772
FIELD LABEL: MESSAGE TEX            DATA TYPE... WORD-PROCESSING

 ┌──────────────────────────────────────────────────────────────┐
 |SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE: YES        |
A|SHALL "|" CHARACTERS IN THIS TEXT BE TREATED LIKE ANY OTHER
CHARACTERS: NO |
 |                                                              |
 └──────────────────────────────────────────────────────────────┘
   WRITE ACCESS:
         SOURCE:
```

```
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... NO

     MANDATORY: NO
   HELP-PROMPT: The text of the incoming messages for this
transmission.
XECUTABLE HELP:
________________________________________________________________



COMMAND:                                         Press <F1>H for
help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. With WORD-PROCESSING-type fields. Fileman asks two questions in the "popup" window:

```
"SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE:"
```

If you answer YES to this question, text is automatically wrapped at word boundaries to fit in the column in which it is being printed. Yes is the default.

```
"SHALL "|" CHARACTERS IN THIS TEXT BE TREATED LIKE ANY OTHER
CHARACTERS:"
```

If you answer NO to this question, the vertical bar (|) character will be ignored. No is the default.

TIP: When it is important that lines of text be printed exactly as they were entered, answer:

- NO to the prompt, "SHALL THIS TEXT NORMALLY APPEAR IN WORD-WRAP MODE:" question, and
- YES to the prompt, "SHALL "|" CHARACTERS IN THIS TEXT BE TREATED LIKE ANY OTHER CHARACTERS:"

COMPUTED Field

In the following example, the DATA TYPE field for the JUST RELEASED field (#1000) in the ORDER file (#100) has a value of COMPUTED:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: JUST RELEASED
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                       Field #1000 in File #100
FIELD LABEL: JUST RELEASED                    DATA TYPE... COMPUTED
  ┌---------------------------------------------------------------┐
  |COMPUTED-FIELD EXPRESSION:                                     |
  |#.68+2>TODAY                                                   |
A |           TYPE OF RESULT: BOOLEAN                             |
  |                       NUMBER OF FRACTIONAL DIGITS TO OUTPUT:  |
  |                              SHOULD VALUE ALWAYS BE ROUNDED:  |
  |     WHEN TOTALLING, SHOULD SUMS BE SUMS OF COMPONENT FIELDS:  |
  |          LENGTH OF FIELD: 3                                   |
  └---------------------------------------------------------------┘

                      IS THIS FIELD MULTIPLE... NO

     MANDATORY: NO
   HELP-PROMPT:
XECUTABLE HELP:

________________________________________________________________
COMMAND:                         Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. With COMPUTED-type fields Fileman displays the characteristics of the computed expression in the "popup" window.

REF: The syntax of these expressions is explained fully in the "Computed Expressions" section.

Field #.68 is the RELEASE DATE/TIME field. The JUST RELEASED field will be TRUE if the RELEASE DATE/TIME was less than two days ago.

NOTE: You can specify this virtual value to be BOOLEAN, STRING-VALUED, DATE-VALUED, or NUMERIC. Only in the last case are the three fields following the "TYPE OF RESULT" prompt editable.

POINTER TO A FILE Field

In the following example, the DATA TYPE field for the WHO ENTERED field (#3) in the ORDER file (#100) has a value of POINTER TO A FILE:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: WHO ENTERED
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                     Field #13  in File #100
FIELD LABEL: WHO ENTERED                   DATA TYPE... POINTER
  ┌-------------------------------------------------------------┐
  |     POINT TO WHICH FILE: NEW PERSON                          |
  |                                                             |
A |     SHALL 'ADDING A NEW FILE ENTRY ("LAYGO") BE ALLPOWED: NO
  |                                                             |
  └-------------------------------------------------------------┘
   WRITE ACCESS:
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... NO

     MANDATORY: NO
   HELP-PROMPT: Enter the name of the person who entered this
```

```
order.
XECUTABLE HELP:
______________________________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. With POINTER TO A FILE-type fields, Fileman asks you to enter the pointed-to file name in the "popup" window. The file that is pointed to (in this case, the NEW PERSON file (#200)) *must* already exist on your system. If you enter a **?** (single question mark) at this prompt, you will be presented with a list of the available files from which you can choose.

By answering NO to the "LAYGO" question, you insure that users who are editing the "WHO ENTERED" data are not able to add a new entry on the fly to the NEW PERSON file.

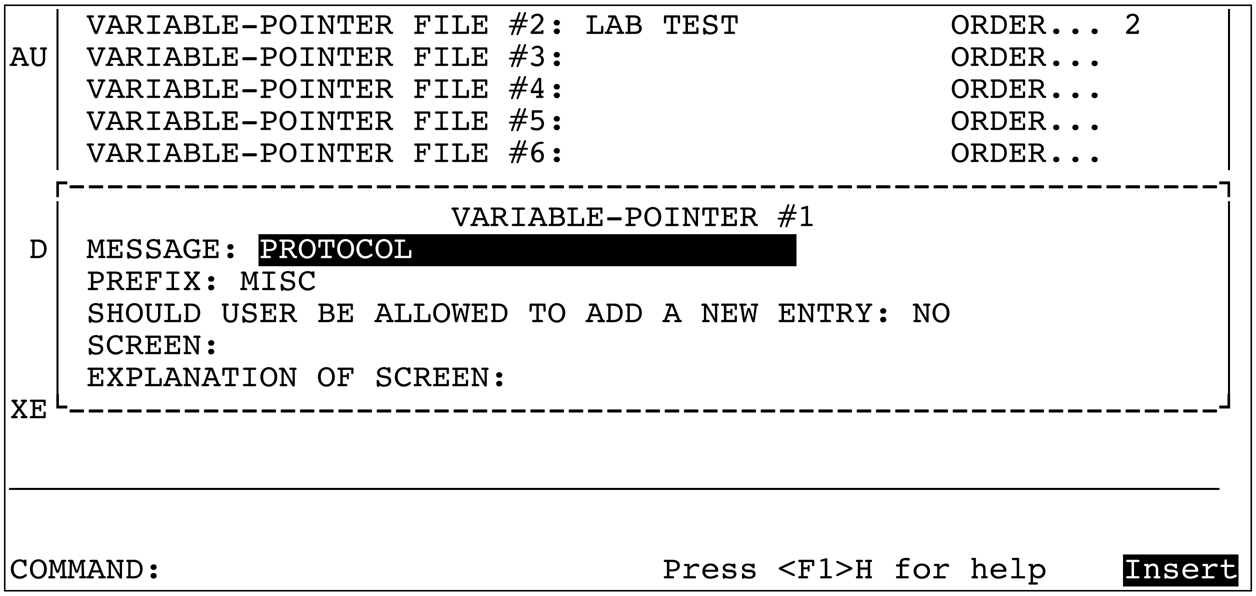
VARIABLE-POINTER Field

In the following example, the DATA TYPE field for the ITEM ORDERED field (#7) in the ORDER file (#100) has a value of VARIABLE-POINTER:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: ITEM ORDERED
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                     Field #7 in File #100
FIELD LABEL: WHO ENTERED           DATA TYPE... VARIABLE-POINTER
  ┌-------------------------------------------------------------┐
  │ VARIABLE-POINTER FILE #1: OPTION             ORDER... 1     │
```



```
  │ VARIABLE-POINTER FILE #2: LAB TEST           ORDER... 2  │
AU│ VARIABLE-POINTER FILE #3:                    ORDER...    │
  │ VARIABLE-POINTER FILE #4:                    ORDER...    │
  │ VARIABLE-POINTER FILE #5:                    ORDER...    │
  │ VARIABLE-POINTER FILE #6:                    ORDER...    │
  ┌----------------------------------------------------------┐
  │                    VARIABLE-POINTER #1                   │
 D│ MESSAGE: PROTOCOL                                        │
  │ PREFIX: MISC                                             │
  │ SHOULD USER BE ALLOWED TO ADD A NEW ENTRY: NO            │
  │ SCREEN:                                                  │
  │ EXPLANATION OF SCREEN:                                   │
XE└----------------------------------------------------------┘

_____________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

In Screen Mode, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. With VARIABLE-POINTER-type fields, Fileman asks you to enter the pointed-to file name and its order in the first "popup" window. In this case, the first VARIABLE-POINTER file entered was the OPTION file, which would already have to exist. The ORDER was also set to one. When you press the **Enter** key at the "ORDER" prompt for each VARIABLE-POINTER, there is an additional "popup" window (i.e., a "popup window within the popup window"). The additional questions pertaining to the VARIABLE-POINTER file you are currently entering will appear in this secondary "popup" window.

As you can see in this example, the OPTION file's MESSAGE, is associated with "PROTOCOL." Since its ORDER is one, it is the first file searched. The PREFIX "MISC" can also be used to refer to the OPTION file. Just as with the WHO ENTERED POINTER TO A FILE field (previously described), users cannot add new options to the OPTION file on the fly when they are entering an ITEM ORDERED in the ORDER file because the "SHOULD USER BE ALLOWED TO ADD A NEW ENTRY:" prompt is NO.

Creating a Multiple

The following example illustrates creating a Multiple field. For this example, we will simulate creating the RESPONSES field (#4.5) in the ORDER file (#100). It has a DATA TYPE of NUMERIC:

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: RESPONSES
  Are you adding 'RESPONSES' as a new FIELD? No// Y
  FIELD NUMBER: 4.5// <Enter>
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                     Field #4.5 in File #100
FIELD LABEL: RESPONSES                   DATA TYPE... NUMERIC

          TITLE:
          AUDIT:
AUDIT CONDITION:
    READ ACCESS:
  DELETE ACCESS:
   WRITE ACCESS:
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... YES
┌--------------------------------------------------------------┐
│ SHOULD USER SEE AN "ADDING A NEW ENTRY" MESSAGE: NO
│
│ HAVING ENTERED OR EDITED ONE MULTIPLE, HSOULD USER BE ASKED
ANOTEHR: NO    │
└--------------------------------------------------------------┘
______________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

A Multiple field is created just as any other, except that the "IS THIS FIELD

MULTIPLE..." question is answered YES. In Screen Mode, after answering YES to this question, a "popup" window appears containing editable attributes pertinent to a Multiple field. With Multiple-type fields, Fileman asks if you want users to be notified when they are adding new entries and if users should be asked if they want to make another entry. In this case, we answered NO to both questions.

Subfields

To create or edit Subfields of a Multiple field, select a Multiple-valued field (e.g., the RESPONSES Multiple field, previously created):

```
Select OPTION: MODify File Attributes
Do you want to use the screen-mode version? YES// <Enter>

MODIFY WHAT FILE: ORDER// <Enter>


Select FIELD: RESPONSES
```

You will then be taken into a Screenman form where you can edit the properties of the field, as shown below:

```
                        Multiple Field #4.5 in File #100

MULTIPLE-FIELD LABEL: RESPONSES
    READ ACCESS:
   WRITE ACCESS:
         SOURCE:



_______________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

In Screen Mode, after entering a Multiple field, a special screen appears that displays information about the Multiple *as a whole*.

NOTE: If you wanted to delete *the entire Multiple field*, you would enter an at-sign ("@") at the "MULTIPLE-FIELD LABEL" prompt.

After viewing this screen, you can proceed to add fields to the Multiple or to edit existing Subfields. This is done at the "Select SUB-FIELD:" prompt that is displayed when you exit Screen Mode (shown below).

Numeric Subfield

After selecting a Multiple-valued field (e.g., the RESPONSES Multiple field in the ORDER file (#100)), you can enter or modify the Multiple's subfields by entering the field's number or name (label) at the "Select *xxxxxxx* SUB-FIELD:" prompt (where *xxxxxxx* represents the name of the Multiple).

A .01 field with the same name as the Multiple field was added automatically when the field was identified as a Multiple. The .01 field is the identifying key for an entry in the Subfile (Multiple); it is similar to a file's .01 field, which is the file's identifying key. In the following example, the .01 subfield is edited to have a DATA TYPE of NUMERIC:

```
Select RESPONSES SUB-FIELD: .01 <Enter>  ITEM ENTRY
```

You will then be taken into a Screenman form where you can edit the properties of the subfield, as shown below:

```
                 Field #.01 in Sub-File #100.045 of File #100
FIELD LABEL: ITEM ENTRY                          DATA TYPE... NUMERIC

  ┌-------------------------------------------------------------┐
  |                INCLUSIVE LOWER BOUND: 1                     |
AU|                INCLUSIVE UPPER BOUND: 9999999               |
  |              IS THIS A DOLLAR AMOUNT: NO                    |
  | MAXIMUM NUMBER OF FRACTIONAL DIGITS: 0                      |
  └-------------------------------------------------------------┘
         SOURCE:
 DESCRIPTION...        TECHNICAL DESCRIPTION...

                      IS THIS FIELD MULTIPLE... YES

     MANDATORY: NO
   HELP-PROMPT: Type a Number between 1 and 9999999, 0 Decimal
Digits
XECUTABLE HELP:
```

```
________________________________________________________________________________


COMMAND:                                 Press <F1>H for help     Insert
```

In Screen Mode, while editing a sub-field of a Multiple, you will notice that the heading at the top of the screen reminds you that you are now editing a field *within a Multiple* (e.g., "Field #.01 in Sub-File #100.045 of File #100").

Also, whenever the DATA TYPE field is editable, a "popup" window appears, containing editable attributes of the field that are pertinent to its specific DATA TYPE field value. With NUMERIC-type fields, Fileman asks you to enter the "INCLUSIVE LOWER BOUND" (e.g., set to 1) and the "INCLUSIVE UPPER BOUND (e.g., set to 9999999). In addition, you are asked if the numeric value is a dollar amount and if decimal digits will be allowed (i.e., the "MAXIMUM NUMBER OF FRACTIONAL DIGITS").

A default help prompt is automatically written for you with the DATA TYPE field of NUMERIC. In this case, the English message "Type a Number between 1 and 9999999, 0 Decimal Digits" has been built automatically from the specifications. As always, you can accept the default help prompt or change it using the Replace ... With syntax.

NOTE: This help information is displayed when the user inputs a single question mark ("**?**") when editing this field.

Chapter 9: File Utilities

Various file utilities are provided as options on Fileman's Utility Functions menu [DIUTILITY].

NOTE: Some additional functionality for modifying files is contained in the separate Modify File Attributes option, which is on the main Fileman menu.

Verify Fields

Located on the Utility Functions menu, the Verify Fields option uses a field's definition to verify the data stored in a file. After invoking this option, you identify the file you want to examine. Then at the "VERIFY WHICH FIELD:" prompt, you can ask for values of a specific field to be checked or you can ask that all fields be verified by entering ALL at the prompt.

The Verify Fields option will report if the value stored in the file is not a valid value based on the field’s data type and input transform. For example:

- A date field must contain a valid Fileman date
- The code of a set of codes field must be one of the specified codes
- A number must be a number within the specified range
- A pointer or variable pointer value must point to an existing entry in the pointed-to file
- If the field has an input transform, the value will be checked against that transform and, if necessary, it will be passed through an existing output transform and checked again against the input transform
- Cross-references are checked to assure that they exist and reference and existing entry
- The length of a cross-referenced is verified to be no greater than 30 characters

Following is a sample report illustrating how the Verify Fields option identifies the field and cross-reference being checked and shows the IEN, entry .01 value, and a brief description of each error found.

```
VERIFY FIELDS REPORT
KBAN BROKEN FILE FILE (#11310003) JAN 09, 2013 14:33 PAGE 1
-----------------------------------------------------------------


--11310003,.01--FIELD #.01 NAME-- (FREE TEXT)
(CHECKING CROSS-REFERENCE)

ENTRY#    NAME      ERROR

6    BAD B XREF          "BAD B XREF" not properly Cross-referenc
10   THIRTY-TWO CHARACTER LIMIT ENT     WRONG "B" CROSS-REF 'BAD
B XREF'
99   99 DANGLING "B" CROSS-REF     'BAD B XREF'
9    LOOPY OUTPUT TRANSFORM   WRONG "B" CROSS-REF 'LOOPY OUTPUT
TRANS'
10   THIRTY-TWO CHARACTER LIMIT ENT     DUPLICATE "B" CROSS-REF
'THIRTY-TWO CHAR


--11310003,.02--FIELD #.02 POINTER-- (POINTER)
(CHECKING CROSS-REFERENCE)

ENTRY#    NAME      ERROR

2    BAD POINTER    No '6666' in pointed-to File
7    BAD C (POINTER) XREF     "4" not properly Cross-referenced
7    BAD C (POINTER) XREF    WRONG "C" CROSS-REF '44'
2    BAD POINTER    WRONG "C" CROSS-REF '7777'


--11310003,.03--FIELD #.03 DATE-- (DATE)
(CHECKING CROSS-REFERENCE)

3    BAD DATE  "ABCDEF" fails Input Transform
8    BAD D (DATE) XREF   "3220801": D index (#1051) not properly
set


--11310003,.04--FIELD #.04 SET-- (SET OF CODES)
```

```
4    BAD SET   "U" not in Set


--11310003,.05--FIELD #.05 NUMBER-- (NUMERIC)

5    BAD NUMBER     "ABCDEF" fails Input Transform


--11310003,.06--FIELD #.06 LOOPY OUTPUT TRANSFORM-- (DATE)
     .
     .
     .
```

If more than one discrepancy is found between the current definition and the data on file, you will be asked: STORE THESE ENTRY ID'S IN TEMPLATE. If you identify a template name at this prompt, the list of those entries containing the inconsistent data will be saved in the template. Later you would be able to "SORT BY:" the entries in this template to display or edit them.

REF: For information on how to execute or avoid executing any part of the INPUT transform when the Verify Fields option is being run, see the "Input Transform" section in the "Advanced File Definition" chapter in the *Fileman Programmer Manual*.

NOTE: Some parts of a field's INPUT transform (whose main purpose is to validate data as a user enters it) may be inappropriate when being executed in the context of the Verify Fields option.

Cross-Reference a Field or File

Traditional Cross-references:

Types of Traditional Cross-references
Edit a Traditional Cross-reference
Create a Traditional Cross-reference
Delete a Traditional Cross-reference

New-Style Cross-references:

Edit a New-Style Cross-reference
Create a New-Style Cross-reference
Delete a New-Style Cross-reference

There are seven types of Traditional cross-references and two types of New-Style cross-references available. Generally, a cross-reference in Fileman specifies that some action is performed when the field's value is entered, changed, or deleted. For several types of cross-references, the action consists of putting the value into a list—an index used when looking up an entry or when sorting. The regular cross-reference is used for sorting and for lookup; you can limit it to sorting only. The KWIC, mnemonic, and SOUNDEX cross-references are also used for lookup.

You can sort a file on any field (except a WORD-PROCESSING-type field) whether or not a cross-reference exists for the field. However, sorting is done more quickly and efficiently if a regular cross-reference exists on the field.

When a file is created, a Traditional cross-reference on the NAME (#.01) field is automatically established. You can add or delete cross-references at any time using the Cross-Reference a Field or File option [DIXREF] located on the Utility Functions menu [DIUTILITY]. This option can also be used to enter a description of a cross-reference and to prevent the cross-reference from being deleted.

You can create a cross-reference on a Multiple field, a Multiple's subfields, or on any other field type except a WORD-PROCESSING-type field. For example, the PATIENT file contains the AGE AT ONSET subfield in the DIAGNOSIS Multiple. If you create a regular cross-reference for a field in a Multiple, you can choose in what context the cross-reference will be used. You might want to cross-reference the whole file by AGE AT ONSET (so that a report sorted by AGE AT ONSET could be produced efficiently). Alternately, you might want to cross-reference only an individual patient's diagnoses by onset age (so that a lookup of diagnosis could be done using AGE AT ONSET).

Types of Traditional Cross-references

Cross-reference	Description
REGULAR	The field value is sorted and stored in the cross-reference. The regular cross-reference is used for sorting. If you wish, it will be used when looking up entries also. The cross-reference that is automatically created on the NAME field (#.01 field) when a file is created is a regular cross-reference; this is the "B" cross-reference.
KWIC	Key Word in Context—each word of three or more letters in the field value becomes a separate cross-reference. A space is considered the primary word separator. For example, KING LEAR can be looked up under either KING or LEAR. Uppercase or lowercase two letter words such as IN, AN, OR, and IS are *not* considered key text. The words THE, AND, THEN, FOR, FROM, OTHER, THAN, WITH, THEIR, SOME, and THIS (upper- or lowercase) are *not* considered key text. Quotation marks are also *not* considered key text. You can also specify that KWIC separates words at most punctuation marks except quotation marks (e.g., KING-LEAR , KING/LEAR, etc., will be found with LEAR). A list of punctuation marks is presented for your selection.
MNEMONIC	The field's values are cross-referenced along with the NAME (#.01) field cross-reference (so that, for example, the MAIDEN NAME field's values are found along with NAME values in any lookup). Typically, the cross-reference on the NAME field is searched first when doing a lookup.

Cross-reference	Description
MUMPS	Those with programmer access can create special cross-references by putting M code into the SET and KILL logic of a cross-reference. You can use the M code entered to accomplish any task that *must* be done when the value in a field is entered, changed, or deleted.
SOUNDEX	The field's value is transformed into a four-character string representing its phonetic properties. That string becomes the cross-reference. For example, soundex transformation would access GONZALEZ, GONZELES, Gonzales, and Gonsalless as equivalents; entry of any one of these forms looks up all the others automatically.
TRIGGER	Whenever the field is updated, a different field can be automatically updated at the same time. **REF:** For more details, see the "Trigger Cross-References" chapter in the *Fileman Programmer Manual*.
BULLETIN	Whenever a field is updated, a MailMan message is sent notifying specified users that an update has occurred. The Bulletin cross-reference is only available when Fileman is installed with MailMan.

Edit a Traditional Cross-reference

To edit a Traditional cross-reference, identify the field or subfield you wish to edit. Fileman will display the type of cross-references on the field and offer you the choices of Edit, Delete, or Create. Select Edit at this prompt and you will have the opportunity to edit or add a No Deletion message and to enter a description of the cross-reference.

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
<Enter>
```

```
MODIFY WHAT FILE: TEST

Select FIELD: .01 <Enter>  NAME

CURRENT CROSS-REFERENCE IS REGULAR 'B' INDEX OF FILE

Choose E (Edit)/D (Delete)/C (Create): E

NO DELETION MESSAGE: NO, DON'T DELETE THIS X-REF!
```

This FREE TEXT message indicates that the cross-reference *cannot* be deleted. As long as a message is retained, the cross-reference cannot be deleted. The user will see this message whenever an attempt is made to delete the cross-reference.

```
DESCRIPTION:
1>Used for look-up on and sorting by name.
2><Enter>
```

The description appears in a standard DD listing.

TIP: It is important to describe cross-references that are unusual or especially critical. Consider describing all MUMPS, trigger, and bulletin cross-references.

Create a Traditional Cross-reference

If you'd like to create a Traditional cross-reference for a field, proceed in the following manner:

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
<Enter>

MODIFY WHAT FILE: TEST

Select FIELD: 1 <Enter>  DATE

NO CURRENT CROSS-REFERENCE
```

```
WANT TO CREATE A NEW CROSS-REFERENCE FOR THIS FIELD?  NO// YES

CROSS-REFERENCE NUMBER:   1// <Enter>

Select TYPE OF INDEXING:  REGULAR// <Enter>
WANT CROSS-REFERENCE TO BE USED FOR LOOKUP AS WELL AS FOR
SORTING?
  YES// <Enter>

NO DELETION MESSAGE: <Enter>
DESCRIPTION:
1>Lookup and sorting can be done by date using this Regular
2>cross-reference.
3><Enter>
```

Delete a Traditional Cross-reference

The following dialogue shows how to delete a Traditional cross-reference:

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
<Enter>

MODIFY WHAT FILE: TEST

Select FIELD: 1 <Enter>  DATE

CURRENT CROSS-REFERENCE IS REGULAR 'C' INDEX OF FILE

Choose E (Edit)/D (Delete)/C (Create): D
Are you sure that you want to delete the CROSS-REFERENCE? NO//
YES
```

New-Style Cross-references

Two types of New-Style cross-references are available: Regular and MUMPS. They are like their Traditional cross-reference counterparts, but New-Style cross-references offer some unique advantages:

Compound Cross-references—You can create not only ***simple cross-references*** that are based on a single field, but ***compound cross-references,***

cross-references that are based on more than one field in a file. For example, in a regular New-Style "C" index you can store both the Name and ID Number of a record as subscripts in a single index:

```
^DIZ(1000,"C","FMPATIENT,25","A56789",14) =
^DIZ(1000,"C","FMPATIENT,10","D1234",5) =
```

In order to create this kind of index with a Traditional cross-reference, you would have to create two MUMPS-type cross-references, one on the NAME field and one on the ID Number field. New-Style cross-references allow you to define this compound cross-reference once as a regular index that Fileman can use for lookup and sorting.

Field- or Record-Level Execution—Since a Traditional cross-reference is defined on a particular field, the action associated with that cross-reference is performed whenever the field is edited. With New-Style cross-references, you can specify that the action associated with a cross-reference be performed only once after the entire record has been edited, typically at the end of the editing session. Record-level execution would normally be selected for compound cross-references.

If the "C" index in the example above were defined using Traditional MUMPS-type cross-references, and both the NAME and ID Number fields were contained in a single INPUT template, the index would be updated when the NAME field was edited, and then again when the ID Number was edited. But if the cross-reference were defined as a New-Style compound index with record-level execution, the index would be updated only once after the entire record was edited, after changes to both the NAME and ID Number fields had been completed.

Code to Kill the Entire Index—This is code that Fileman can execute to remove an entire index from a file. This can make re-indexing a file much more efficient. To delete an index, Fileman can execute the *Kill Entire Index Code*, instead of looping through all the record in a file and removing each record's index one at a time.

Activity—New-Style cross-references can have an Activity of "**R**" and/or

"**I**" to allow you to control whether the cross-reference should be fired during **R**eindexing and/or **I**nstallation (KIDS). If you call IX^DIK, IX1^DIK, or IXALL^DIK or if you select the Re-Index File option [DIRDEX] located on Fileman's Utility Functions menu [DIUTILITY] to re-index all cross-references, only those New-Style cross-references that contain an "**R**" in Activity will be fired.

If you explicitly select a cross-reference in an EN^DIK, EN1^DIK, or ENALL^DIK call or in the Re-Index File option [DIRDEX] on Fileman's Utility Functions menu [DIUTILITY], that cross-reference is fired regardless of its Activity. Also, when a field is edited, Fileman ignores Activity and fires all cross-references on that field; though, you can control whether a cross-reference is fired by entering Set and Kill Conditions.

REF: For more information on the Re-Index File option and limiting re-indexing on some files, see the "Re-Index File" and "Limits on Re-indexing Files" sections.

Collation—You can specify forwards or backwards collation, the direction in which Fileman's lookup utilities loop through a subscript in an index when entries are returned or displayed to the user. This is especially useful for dates. Developers can store dates in their natural internal Fileman date format, and still display entries in the date index in reverse date order.

Lookup Prompt—Each subscript on an index in the new INDEX file can be assigned a LOOKUP PROMPT. This prompt will be used as the prompt for entry of the lookup value during classic Fileman lookup ^DIC calls. If not filled in, Fileman will default to use the name of the field for that subscript value, if there is one.

Computed Values—Those with programmer access can have any value in the cross-reference be computed; that is, the value is determined from M code that sets the variable X.

Subscript Transforms—Those with programmer access can define a *Transform for Storage* and a *Transform for Display* on subscripts in an index.

The *Transform for Storage* is code that transforms the internal value of a field before it is stored as a subscript in the index. The *Transform for Display* is code that transforms the value stored in the index back to a form that can be displayed to the user.

SET and Kill Conditions—Those with programmer access can enter M code that specifies whether the set or kill logic is fired. The M code sets the variable X to Boolean true only if the logic should be executed. The "before" and "after" values are available in the X, X1, and X2 arrays.

The X, X1, and X2 Arrays—Those with programmer access can reference the X, X1, and X2 arrays in the SET and KILL logic and the SET and KILL conditions of New-Style cross-references. When a field is edited and the cross-reference logic is executed, the field's corresponding X1 array element contains the old value of the field, the X2 array element contains the new value of the field, and the X array element contains either the old or new value, depending on whether the SET logic, SET condition, KILL logic, or KILL condition is being executed:

Array	Value in KILL Logic/KILL Condition	Value in SET Logic/SET Condition
X(order#)	Old value	New value
X1(order#)	Old value	Old value
X2(order#)	New value	New value

The variables X, X1, and X2 always equal X(1), X1(1), and X2(1), respectively.

If an order number in the cross-reference refers to the .01 field, X1(order#) is set to null when the SET logic and SET condition are executed during record creation. Similarly, X2(order#) is set to null when the KILL logic or condition are executed during record deletion.

Key Support—A regular New-Style index can be used as the Uniqueness

Index for a key. Fileman ensures that all fields in a Uniqueness Index have values (are not null), and that those values, taken collectively, are unique across all records in the file.

REF: For more information on keys and how to create them, see the "Key Definition" section.

Edit a New-Style Cross-reference

To edit a New-Style cross-reference, identify the file or subfile you wish to edit. Fileman will display the cross-references on the file and offer you the choices of Edit, Delete, or Create:

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
NEW
MODIFY WHAT FILE: TEST
Select Subfile: <Enter>

Current Indexes on file #16026:
  75     'COMP' index
  85     'XR202' index
  106    'H' index
  116    'AC' index
  141    'C' index

Choose E (Edit)/D (Delete)/C (Create): EDIT

Which Index do you wish to edit? C
```

NOTE: The numbers displayed to the left of each cross-reference are the Internal Entry Number of the cross-reference stored in the INDEX file.

You will then be taken into a Screenman form where you can edit the properties of the New-Style cross-reference, as shown below:

```
Number: 106                           EDIT AN
INDEX                           Page 1 of 2
-------------------------------------------------------------------
```

```
        File: 16026                                       Root File:
16026
  Index Name: H                                           Root Type: INDEX
FILE

Short Description:
TEST
 Description (wp):    (empty)

        Type: REGULAR

    Activity: IR
   Execution: FIELD

         Use: LOOKUP & SORTING

         Do Not ReIndex: NO RE-INDEXING ALLOWED
________________________________________________________________


COMMAND:                                 Press <F1>H for help    Insert
```

Alternatively, this field can be null/blank, see Figure 173.

NOTE: For additional help, enter a single question mark ("**?**") or two question marks ("**??**") at any prompt.

Create a New-Style Cross-reference

To create a New-Style cross-reference, proceed in the following manner:

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
NEW
MODIFY WHAT FILE: TEST
Select Subfile: <Enter>

Current Indexes on file #16026:
  75     'COMP' index
  85     'XR202' index
  106    'H' index
  116    'AC' index
  141    'C' index
```

```
Choose E (Edit)/D (Delete)/C (Create): CREATE
Want to create a new Index for this file? No// YES

Type of index: REGULAR// <Enter>

Want index to be used for Lookup & Sorting
  or Sorting Only: LOOKUP & SORTING// <Enter>

Index Name: J// <Enter>
```

NOTE: The numbers displayed to the left of each cross-reference are the Internal Entry Number of the cross-reference stored in the INDEX file.

You will then be taken into a Screenman form where you can edit the properties of the New-Style cross-reference, as shown below:

```
Number: 142                                 EDIT AN
INDEX                          Page 1 of 2
-------------------------------------------------------------------

        File: 16026                                Root File:
16026
  Index Name: J                                    Root Type: INDEX
FILE

Short Description:
TEST
 Description (wp):    (empty)

        Type: REGULAR

    Activity: IR
   Execution: FIELD

         Use: LOOKUP & SORTING

         Do Not ReIndex:

___________________________________________________________________


COMMAND:                                  Press <F1>H for help    Insert
```

Alternatively, this field can indicate "NO RE-INDEXING ALLOWED", see Figure 171.

NOTE: For additional help, enter a single question mark ("**?**") or two question marks ("**??**") at any prompt.

Delete a New-Style Cross-reference

The following dialogue shows you how to delete a New-Style cross-reference:

```
Select OPTION: UTILITY FUNCTIONS
Select Utility Functions Option: CROSS-REFERENCE A FIELD OR FILE

What type of cross-reference (Traditional or New)? Traditional//
NEW
MODIFY WHAT FILE: TEST
Select Subfile: <Enter>

Current Indexes on file #16026:
  75     'COMP' index
  85     'XR202' index
  106    'H' index
  116    'AC' index
  141    'C' index

Choose E (Edit)/D (Delete)/C (Create): DELETE

Which Index do you wish to delete? 141 <Enter>  C
Are you sure you want to delete the Index? No// YES

  Index definition deleted.
  Removing old index ...  DONE!

Press RETURN to continue: <Enter>
```

Identifier

NOTE: If you want to uniquely identify an entry in your file by a combination of fields, and to force that uniqueness, then you will most likely want to create a KEY on your file, rather than using Identifier fields (which do not force the uniqueness). If a field is part of the PRIMARY KEY for a file, then it should not be marked as an Identifier as well.

REF: For more information on creating a KEY, see the "Key Definition" section.

An identifier is a designation you can give to a field that you want permanently associated with the .01 field (NAME) of a file. The SSN field of our PATIENT file example has been defined as an identifier field. Each time a patient's entry is referenced, the SSN will be displayed to help positively identify the entry. When a new entry is added to the file, the user will be asked to provide the SSN.

A field that is not multiple-valued can be specified as an identifier for a file simply by using the Identifier option available on the Utility Functions menu [DIUTILITY].

A multiple-valued field *cannot* be designated as an identifier, however, a subfield of that multiple-valued field *can* be designated as an identifier for the multiple. The DIAGNOSIS field in the PATIENT file cannot, for example, be designated as an identifier, but its subfield AGE AT ONSET can be designated as an identifier of DIAGNOSIS. This feature is discussed in more detail later in this section.

These are the steps for setting up the sample SSN field as an identifier:

```
MODIFY WHAT FILE: PATIENT
Select FIELD: SSN
Want to make 'SSN' an identifier? NO// Y
Want to display SSN whenever a lookup is done
     on an entry in the 'PATIENT' file?  YES // <Enter>
```

Here, the positive answer to the last question causes the patient's SSN value to show up whenever a lookup on a patient is done. For example:

```
Select PATIENT NAME: FMPATIENT
    1  FMPATIENT,25   000223333
    2  FMPATIENT,29   000114444
CHOOSE 1-2:
```

An identifier field will not be asked if its WRITE access security does not match the Fileman Access Code of the user. If the identifier field has been specified (in the Modify File Attributes option) as a required field, the user *must* type a valid answer to its prompt when it is asked as an identifier; otherwise, the entry just created is deleted.

Using the caret key ("^") for jumping is not allowed for identifier fields in the Enter or Edit File Entries option when adding a new entry. If you attempt to use the caret key in a field designated as an identifier in an edit session, the entry just created is deleted. Since the SSN field in our example is mandatory and an identifier, this ensures that every patient in our PATIENT file will have an SSN recorded.

As mentioned above, you could make the AGE AT ONSET subfield an identifier for the multiple DIAGNOSIS field as follows:

```
Select UTILITY OPTION: IDENTIFIER
Select FIELD: DIAGNOSIS     (multiple)
Select DIAGNOSIS SUB-FIELD: AGE AT ONSET
Want to make 'AGE AT ONSET' an Identifier? NO// Y
Want to display AGE AT ONSET whenever a lookup is done
     on an entry in the 'DIAGNOSIS' file? YES// N
```

As a result of this dialogue, every time a new DIAGNOSIS for a patient is entered, the AGE AT ONSET would be asked. The AGE AT ONSET would not, however, be automatically displayed at subsequent DIAGNOSIS lookups.

To drop a field's status as an identifier, simply return to the Identifier option, select the field, and answer YES to the question:

```
Field is already an Identifier; want to delete it?  NO// Y
```

Re-Index File

Use the Re-Index File option [DIRDEX] when you create a new cross-reference on a field that already contains data and you want to re-index the file. The following dialogue is presented when re-indexing a file:

```
DO YOU WISH TO RE-CROSS-REFERENCE ONE PARTICULAR INDEX? NO//
<Enter>
```

```
OK, ARE YOU SURE YOU WANT TO KILL OFF THE EXISTING INDEX? NO// Y
DO YOU THEN WANT TO 'RE-CROSS-REFERENCE'? YES// <Enter>
```

All the cross-references for the file will be fired except for bulletins. This dialogue will execute triggers and MUMPS cross-references.

If a file contains more than one cross-reference, you can get a list of them by entering a single question mark ("**?**") in response to the "DO YOU WISH TO RE-CROSS-REFERENCE ONE PARTICULAR INDEX?" prompt. You can then re-index a single cross-reference or all of the file's cross-references.

Limits on Re-indexing Files

There are some files that should *not* be re-indexed! When those files are inadvertently re-indexed, it can cause major problems, and necessitate restores from backups. Currently, there is no way to prevent such files from being re-indexed, except for putting "DO NOT RE-INDEX" in the file description, which is ineffective. The Re-Index File option [DIRDEX] is a powerful, useful tool, but it can cause a lot of damage when one is not paying attention.

Patch DI*22*167 allows individual cross-references to be marked "Do Not Re-Index," and the Re-Index File option [DIRDEX] will respect that. APIs that perform re-indexing will also respect that, with the following exceptions:

APIs that re-index a single record will ignore the no-re-index restriction, which includes the following APIs:

EN^DIK
EN1^DIK
EN2^DIK
IX^DIK
IX1^DIK
IX2^DIK

A cross-reference will be re-indexed if it is specifically named in an API call, regardless of whether it is marked "Do Not Re-Index," which includes the following APIs:

ENALL^DIK
ENALL2^DIK

REF: For more information on these APIs, see the *Fileman Programmer Manual*.

Traditional regular cross-references (e.g., B and C cross-references) will always be re-indexed, and *cannot* be marked "Do Not Re-Index." All other cross-references can be marked "Do Not Re-Index," because re-indexing them might cause problems.

The following cross-reference types can be marked "Do Not Re-Index:"

All New-Style cross-references
Bulletin cross-references
MUMPS cross-references
Trigger cross-references

To mark a cross-reference "Do Not Re-Index," use the Cross-Reference a Field or File option [DIXREF] under the Utility Functions menu [DIUTILITY].

CAUTION: "Do Not Re-Index" can only be undone by KILLing the "NOREINDEX" node in the DD.

REF: For more information on the Cross-Reference a Field or File option, see the "Cross-Reference a Field or File" section.

INPUT Transform (Syntax)

If you have programmer access, you will be able to edit a field's INPUT transform or syntax checker.

REF: For a detailed description of the INPUT transform, see the "Input Transforms" section in the *Fileman Programmer Manual*.

Edit File

The Edit File option available on the Fileman Utility Functions menu [DIUTILITY] displays the various attributes of a file you specify in Screen Mode (i.e., invokes a Screenman form).

Here is an example using the Edit File option with the ORDER file (#100) in Screen Mode:

```
Select Fileman Option: UTILITY Functions

          Verify Fields
          Cross-Reference A Field
          Identifier
          Re-Index File
          Input Transform (Syntax)
          Edit File
          Output Transform
          Template Edit
          Uneditable Data
          Mandatory/Required Field Check
          Key Definition

Select Utility Functions Option: EDIT File

MODIFY WHAT FILE: ORDER// <Enter>
Do you want to use the screen-mode version? YES// <Enter>
```

You will then be taken into a Screenman form where you can edit the properties of the file, as shown below:

```
  FILE NAME: ORDER
                    DESCRIPTION...                 (File # 100)
           Select APPLICATION GROUP:
                        DEVELOPER:
             DATA DICTIONARY ACCESS: #
                      READ ACCESS: #
                     WRITE ACCESS: #
                    DELETE ACCESS: #
                     LAYGO ACCESS: #
                     AUDIT ACCESS: #
                         DD AUDIT: NO
  ASK 'OK' WHEN LOOKING UP AN ENTRY: NO
              FILE SCREEN:
    POST-SELECTION ACTION:
          LOOK-UP PROGRAM:
  CROSS-REFERENCE ROUTINE: ORD2
_______________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

You can use the Edit File option to:

- **Edit the Name of a File—**Edit the file name at the "FILE NAME:" prompt.
- **Delete a File—**If you enter an at-sign ("**@**") at the "FILE NAME:" prompt, you are given the choice of deleting the ***entire file*** and its data attribute dictionary (including all of its templates and file definitions) or just deleting the ***current individual entries in the file***. However, you *cannot* delete a file that is pointed to by another file.
- **Enter or Edit the Description of a File—**You can enter or edit the word processing text description for documenting the file at the "DESCRIPTION..." prompt. This description appears in the Standard, Modified Standard, and Global Map format data dictionary listings.
- **Enter or Edit the Application Group—**You can enter or edit the Application Group at the "Select APPLICATION GROUP:" prompt. Enter a namespace (from two to four characters) indicating a package accessing this file.
- **Enter or Edit the Developer's Name—**You can enter or edit the name of the package developer at the "DEVELOPER:" prompt. Entering two question marks ("**??**") lets you choose from a list of names.
- **Enter or Edit the File Access Parameters—**You can enter or edit the security access to a file by making entries at the "DATA DICTIONARY ACCESS:", "READ ACCESS:", "WRITE ACCESS:", "DELETE ACCESS:", "LAYGO ACCESS:", and/or "AUDIT ACCESS:" prompts.

- **Ask/Do Not Ask Users to Confirm Their Entry Selection—**If you want users who select an entry in a file (for any lookup purpose) to confirm their entry selection by answering positively at the "…OK?" prompt, answer YES at the "ASK 'OK' WHEN LOOKING UP AN ENTRY:" prompt. If you do not want users to confirm their entry selection, answer NO at the "ASK 'OK' WHEN LOOKING UP AN ENTRY:" prompt. The default is NO.
  TIP: Use this feature on files containing many similar or confusingly named entries (e.g., files for drugs).
- **Enter a File Screen—**A line of MUMPS code can be entered here. It should set the $T switch TRUE or FALSE. At the time of execution 'Y' is the number of a File entry, which we want to FILTER for lookup. Thus this code is a 'permanent DIC("S")' for the File.
  Misuse of this can disenable the file!
- For example, this is the file screen for the NEW PERSON file: I $ $SCR200^XUSER.
- **Enter or Edit a Post-Selection Action (only available when you have programmer access)—**If you have programmer access, you can write M code for a Post-Selection Action, for entries in this file.
  REF: Post-Selection Action is explained in the *Fileman Programmer Manual*.
- **Enter a Lookup Routine (only available when you have programmer access)—**If you have programmer access, you also can enter an existing lookup routine. To do this, enter a routine namespace (from three to six characters, no "**^**") at the "LOOK-UP PROGRAM:" prompt. The name you choose for the lookup routine *must* be a routine currently on the system. This special lookup routine will be executed instead of the standard Fileman lookup logic, whenever a call is made to ^DIC.
- **Specify that Cross-references on a File Should be Compiled (only available when you have programmer access)—**If you have programmer access, you also can specify that cross-references on a file should be compiled. To do this, enter a routine namespace (from three to six characters, no "**^**") at the "CROSS-REFERENCE ROUTINE:" prompt. This will become the namespace of the compiled routine(s). If a *new* routine name is entered, but the cross-references

are not compiled at this time, the routine name will be automatically deleted.
- To **stop** the use of the compiled cross-references, enter an at-sign ("@") at the "CROSS-REFERENCE ROUTINE:" prompt. At this point, the cross-references are considered uncompiled, and Fileman will not use the routine for re-indexing. If you decide later to recompile the cross-references, you will be shown the routine name previously used so that you can easily reuse the same routine name. Stopping the use of the compiled cross-reference does *not* delete the compiled routines. If you want, you can delete those routines manually.

OUTPUT Transform

Sometimes, you might want to display a field differently from the way in which it is stored. For example, a Social Security Number can be entered and stored as nine digits, but you may want it to always be displayed with punctuating hyphens. The Output Transform option allows you to make this kind of specification by associating with any field a computed expression that operates on the value of that field

REF: For details about using M code in an OUTPUT transform, see the "OUTPUT Transform" topic in the "Advanced File Definition" chapters in the "Developer Tools" section in the *Fileman Programmer Manual*.

In the dialogue that follows, you'll encounter the responses that you would enter if you want your SSN field to always appear with inserted dashes.

```
Select OPTION: UTILITIES

Select UTILITY OPTION: OUTPUT TRANSFORM

MODIFY WHAT FILE: PATIENT
Select FIELD: SSN
SSN OUTPUT TRANSFORM:
$E(SSN,1,3)_"-"_$E(SSN,4,5)_"-"_$E(SSN,6,9)
```

Remember the following:

The transform does *not* apply when you are inputting data—thus, do not enter the dashes when using the Enter or Edit File Entries option.

To retrieve the internal, stored value of a field that has an OUTPUT transform, you can refer to the INTERNAL(SSN) function.

The internal form of the date is automatically invoked when you are sorting by a DATE/TIME valued field.

Uneditable Data

The Uneditable Data option allows you to specify that a field's data value cannot be edited or deleted by a user. This restriction can be applied to all data types including word-processing fields.

If editing is attempted, the field's value, along with a No Editing message, will be displayed. If, however, the value is part of a subfield, the deletion of the entire entry in the Multiple-valued field will be allowed—unless the .01 field of the Multiple itself is made uneditable.

You can also use this option to remove the uneditable restriction on a field.

Mandatory/Required Field Check

The Mandatory/Required Field Check option checks that fields that are key fields or designated as required contain data. It can check one, a series, or all required entries in a file. If an entry lacks data in a required or key field, a report like the following is furnished:

```
Required-Field-Check  File: 16026   ZZPATIENT           PAGE 1
Entry                              DD-Number/Path    Field
-------------------------------------------------------------
3        FMPATIENT,25                  16026           SSN
                                   DIZ(16026,3
```

If all required and key fields contain data, then the NO REQUIRED FIELD IS MISSING message is displayed.

You can store the results in a template.

Key Definition

Using the Key Definition option, Fileman allows you to define keys on a file or subfile. A key is a group of fields that, taken collectively, uniquely identifies a record. All fields in a key *must* have values (*must* not be null) and those values, taken together, *must* be unique across all records in the file or subfile. Fileman enforces KEY INTEGRITY whenever records are added or edited.

Exactly one key in a file *must* be designated the PRIMARY KEY. All other keys are SECONDARY KEYs. While Fileman enforces the integrity of both primary and SECONDARY KEYs, the PRIMARY KEY is Fileman's principal means of looking up entries in the file. Fileman will prompt for lookup values for each of the PRIMARY KEY fields, and will consider a record a match only if it matches all of the lookup values. The .01 field should be a part of the PRIMARY KEY.

Keys are also useful when transporting data to another system using the Kernel Installation and Distribution (KIDS) system. Since the key fields uniquely identify a record, it is easy to decide whether a record being brought in to the target system needs to be merged to a record that already exists, or whether it is a new record.

Associated with each key is a *Uniqueness Index*, a regular, New-Style cross-reference. The Uniqueness Index helps Fileman enforce KEY INTEGRITY and is used during lookup. When you create a new key, you can have Fileman create a new Uniqueness Index automatically for you, or you can select an existing index to be the Uniqueness Index of the key. The index you select, though, *must meet the following criteria*:

- It *must* be a regular, New-Style cross-reference.
- It *must* be used for lookup and sorting; that is, it cannot have a name that starts with the letter "A".
- It cannot have any set or kill conditions.
- It *must* consist of only field-type cross-reference values, all of which are used as subscripts; that is, it can contain no computed values.
- No subscripts can have transforms.

Create a Key

To create a key, proceed in the following manner:

```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: KEY DEFINITION

MODIFY WHAT FILE: ZZPATIENT// <Enter>
Select Subfile: <Enter>

There are no Keys defined on file #16026.
Want to create a new Key for this file? No// YES

Enter a Name for the new Key: A// <Enter>
  Creating new Key 'A' ...
```

You will then be taken into Screen Mode (i.e., Screenman form) where you can edit the properties of the key. Enter a single question mark ("**?**") or two question marks ("**??**") at any prompt for additional help.

```
Number: 5                     EDIT A KEY            Page 1 of 1
-------------------------------------------------------------------
File: 16026                   Name: A           Priority:
PRIMARY

KEY FIELDS:
==========
Field                Seq No.  File         Field Name
-----                -------  ----         ----------


Uniqueness Index:

Index Details...
```

```
________________________________________________________________________


COMMAND:                          Press <F1>H for help    Insert
```

On this screen, in the KEY FIELDS section, you can select the fields you wish to include in this key, and assign each field a sequence number. The sequence number determines the order in which the fields appear as subscripts in the Uniqueness Index. If you select the key fields in this manner, leave the Uniqueness Index field blank. When you exit the form, Fileman will prompt you for a name for the Uniqueness Index, and then create the index automatically for you.

```
I'm going to create a new Uniqueness Index to support Key 'A' of
File #16026.

Index Name: C// <Enter>

  One moment please ...
  Building new index ...  DONE!

Press RETURN to continue:
```

Alternatively, you can leave the information in the KEY FIELDS section blank, and select an existing Uniqueness Index. When you exit the form, Fileman checks that the information in the KEY FIELDS section is consistent with the selected Uniqueness Index. If there is a conflict, you are asked for a method to resolve the conflict. In this case, select Option #2, "Make Key match Uniqueness Index," as shown below:

```
The Key fields and the fields in the Uniqueness Index don't
match.

     Select one of the following:

          1         Re-Edit the Key
          2         Make Key match Uniqueness Index (also
selected on up-arrow)

Enter response: 2 <Enter>  Make Key match Uniqueness Index (also
selected on up-arrow)
```

```
  Modifying fields in Key ...  DONE!
```

Edit a Key

To edit a key, identify the file or subfile you wish to edit. Fileman will display the cross-references on the file and offer you the choices of Edit, Delete, or Create.

```
Select OPTION:  UTILITY FUNCTIONS
Select UTILITY OPTION: KEY DEFINITION

MODIFY WHAT FILE: ZZPATIENT// <Enter>
Select Subfile: <Enter>

Keys defined on file #16026:

  A  PRIMARY KEY    Uniqueness Index: C
         Field(s):  1) NAME (#.01)
                    2) SSN (#.02)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): EDIT

Which Key do you wish to edit? A// <Enter>
```

You will then be taken into Screen Mode (i.e., Screenman form) where you can edit the properties of the key. Enter a single question mark ("**?**") or two question marks ("**??**") at any prompt for additional help.

Delete a Key

The following dialogue shows how to delete a key.

NOTE: You are also given the option of deleting the Uniqueness Index of the key.

```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: KEY DEFINITION

MODIFY WHAT FILE: ZZPATIENT// <Enter>
Select Subfile: <Enter>
```

```
Keys defined on file #16026:

  A  PRIMARY KEY    Uniqueness Index: C
         Field(s):  1) NAME (#.01)
                    2) SSN (#.02)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): DELETE

Which Key do you wish to delete? A// <Enter>
Are you sure you want to delete the Key? No// Y

  Key 'A' of File #16026 deleted.

Do you want to delete the 'C' Uniqueness Index (#6) on File
#16026 previously
used by Key 'A' of File #16026? YES

  Index definition deleted.
  Removing old index ...  DONE!
```

Verify a Key

When you verify the integrity of a key, Fileman checks that all fields in the key have values (are not null), and that those field values, taken together, are unique across all records in the file. Any problems are reported. You can also save the entries that violate KEY INTEGRITY in a template.

```
Select OPTION: UTILITY FUNCTIONS
Select UTILITY OPTION: KEY DEFINITION

MODIFY WHAT FILE: ZZPATIENT// <Enter>
Select Subfile: <Enter>

Keys defined on file #16026:

  A  PRIMARY KEY    Uniqueness Index: KEYA
         Field(s):  1) NAME (#.01)
                    2) SSN (#.02)

Choose V (Verify)/E (Edit)/D (Delete)/C (Create): V <Enter>
ERIFY

Which Key do you wish to verify? A// <Enter>
STORE THESE ENTRY ID'S IN TEMPLATE: <Enter>
```

```
DEVICE: HOME// <Enter>  Telnet terminal

KEY INTEGRITY CHECK               DEC 31, 1998  09:23    PAGE 1
----------------------------------------------------------------
             Key: A (#5), File #16026
Uniqueness Index: KEYA (#6)

ENTRY #  NAME                   ERROR
-------  ----                   -----
1        FMPATIENT,10             Duplicate Key A (#5)

2        FMPATIENT,10             Duplicate Key A (#5)

3        FMPATIENT,25               Missing Key Field(s):
                                  SSN [16026,.02]
```

In this example, records #1 and #2 have the same key, and record #3 is missing a value for SSN (field #.02).

Moving Data Around

Chapter 10: Import and Export Tools

If you want to use an application like Microsoft Excel to manipulate data stored in a Fileman file, you need some way to exchange that data between Fileman and your application. Fileman provides the Import and Export Tools for this purpose.

Suppose, for example, that you want to use Microsoft Word's Print Merge utility to print a form letter to a list of recipients that is maintained in a Fileman file. You can use Fileman's **Export Tool** to export the list of recipients from the Fileman file to Microsoft Word. Once you have done this, you can use Word to generate your form letters based on the exported list.

What Applications Can You Exchange Data With?

In theory, you can exchange data with any application that supports delimited or fixed-length ASCII data exchange. Many applications do, using a variety of formats. Typically, you can expect the ability to import and export data with the following types of applications:

- Databases
- Spreadsheets
- Statistical and Analysis Programs (SAS, SPSS, etc.)
- Vertical Applications
- Word Processor (data records, *not* word processing text)

NOTE: You can export data records to a word processor, which often uses data records for functions such as print merges. You *cannot* use the

Import or Export Tools to exchange word processing fields from Fileman files, however.

How Data is Moved between Applications

Movement of data between applications that do not "speak the same language" is a complex process because it involves coordinating activities in different computer applications and often in multiple computing environments.

Fileman's **Import** and **Export Tools** use ASCII data exchange. It is the oldest and most widely supported way of exchanging data between applications. Data for a particular record or group of records can be transported in one of two standard formats:

- Delimited
- Fixed-length

To ***export*** data from a Fileman file, use the **Export Tool** to create an ASCII data file containing exported records. The exported data is formatted in such a way that it can be recognized by the particular foreign application. The ASCII data file can then be imported into the foreign application.

To ***import*** data to a Fileman file, use your foreign application to generate an ASCII data file containing records in either delimited or fixed-length formats. Then use the **Import Tool** to load those records into the Fileman file you specify.

Dependency on Correct Data Communication

For import or export of data to succeed, the data *must* be passed correctly on all communication pathways between Fileman and the foreign application. A glitch in the communication of data can cause data exchange to fail.

For example, suppose the foreign application expects the fields in records you are exporting to be separated (or "delimited") by the Tab character (**<TAB>**). The Export Tool can output a **<TAB>** between each field's data value. However, if you use a communication program's screen capture facility to create a file of the exported data and if that communication program automatically changes **<TAB>**s into a certain number of spaces to align text, the exported data will be corrupted and the import will fail.

You should be familiar with your importing or exporting application and with any communications programs that you are using. Knowledge of all the applications involved, starting with Fileman and its Import and Export Tools, increases the likelihood of a successful transfer of data.

Data Formats

Delimited Data Format

Suppose you have a record with LASTNAME = "FMPATIENT", FIRSTNAME = "ONE", AGE = "36". In delimited data format, you choose a delimiter character to place between field values. Let's use a comma as the delimiter character.

A comma (",") is then inserted between each of a record's fields, to "delimit" them. The resulting record, exported in comma-delimited format, would look like:

```
    FMPATIENT,ONE,36
```

Groups of records are exported line-by-line, one line after another. A file of records in comma-delimited format might look like:

```
    FMPATIENT,TWO,1 GREEN LANE,,,Amherst,NH,03012
    FMPATIENT,THREE,30 Plaza Court,,,San Francisco,CA,94105
    FMPATIENT,FOUR,90 123rd St.,,,San Francisco,CA,94112
```

In order to use delimited data format, both applications (the exporting application and the importing application) *must* be able to recognize the format.

Quoted Fields in Delimited Format

Now suppose in the previous example that instead of two separate fields for LASTNAME and FIRSTNAME, there is only a single NAME field for both. Suppose that incoming data you want to place in the single NAME field comes in the form FMPATIENT,FOUR, but you still want to use commas as your delimiter. You can use the "Fields Quoted" setting in the Import form (or the Quote Non-Numeric Fields setting in a Foreign Format) to ignore the delimiter if it is between quotes in the incoming data.

Thus, if you set "Fields Quoted" to YES in your import form, and you pass in a record that looks like:

```
    "FMPATIENT,FOUR",90 123rd St.,,,San Francisco,CA,94112
```

For quoted fields, like "FMPATIENT,FOUR", the Import Tool will ignore the comma delimiter in-between the quotes and treat "FMPATIENT,FOUR" as a single field value.

Fixed-Length Data Format

In fixed-length data format, a standard width is expected for each field in the record. Suppose, for example, you have a record with LASTNAME = "FMPATIENT", FIRSTNAME = "ONE", AGE = "36". 25 characters might be set aside for LASTNAME, 20 characters for FIRSTNAME, and 3 characters for AGE. The resulting record, exported in fixed-length format, would look like:

```
    FMPATIENT                     ONE                36
```

Groups of records are exported line-by-line, one line after another. A file of records in fixed-length format might look like:

```
    FMPATIENT                 TWO                 29
    FMPATIENT                 THREE               47
    FMPATIENT                 FOUR                38
```

In order to use fixed-length data format, both applications (the exporting application and the importing application) *must* be able to recognize the format.

How to Export Data

The following menu shows the options used to export data:

```
  Fileman ...[DIUSER]
    Other Options ...[DIOTHER]
      Data Export to Foreign Format ... [DDXP EXPORT MENU]
        Define Foreign File Format [DDXP DEFINE FORMAT]
            **> Locked with DDXP-DEFINE
        Select Fields for Export  [DDXP SELECT EXPORT FIELDS]
        Create Export Template  [DDXP CREATE EXPORT TEMPLATE]
        Export Data   [DDXP EXPORT DATA]
        Print Format Documentation   [DDXP FORMAT DOCUMENTATION]
```

If you know how to print file entries, you already know most of the procedures to export file entries. The Export Tool is based on the standard Fileman Print File Entries option.

REF: For more information on the Print File Entries option, see the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

The Export Tool creates a specially formatted print output. Some limitations apply to data exports that do not apply to setting up a regular print (e.g., WORD-PROCESSING-type fields cannot be exported). Some capabilities are available when exporting that are not when you are printing (e.g., the records you export can be longer than 245 characters, if you are using a delimited format—see the description of the Maximum Output Length FOREIGN FORMAT attribute below). These differences are discussed below.

The steps to export data are:

1. Make sure there is a FOREIGN FORMAT file (#.44) entry available to export your data in the format expected by the receiving application.
2. Select the fields you want to export (Select Fields for Export option). This creates a SELECTED EXPORT FIELDS template.
3. Create an EXPORT Template. This is where you combine the SELECTED EXPORT FIELDS template with a desired FOREIGN FORMAT.
4. Export Data option. This is where you select which entries to export, and perform the export.

Make Sure a FOREIGN FORMAT File Entry is Available

First, you need to determine an ASCII data format (some form of delimited or fixed-length) that your foreign application will recognize. This is the format you will need the Export Tool to generate.

This data format *must* be set up in advance, as an entry in the FOREIGN FORMAT file (#.44). The following are the major format parameters stored in a FOREIGN FORMAT file (#.44) entry:

- What delimiters are used between fields?
- Does the export use fixed length fields?
- What headers to output before the body of the data, and what footers after the data
- Any special formatting for specific DATA TYPE field values (e.g., dates and numbers)?

Some formats are already set up in advance in the FOREIGN FORMAT file (#.44), targeted towards specific foreign applications. These include:

- Word Data File (Comma)
- Excel (Comma)
- Excel (Tab)
- 1-2-3 Import Numbers

- 1-2-3 Data Parse
- Oracle (Delimited)

Keep in mind that applications are often updated. A format that worked for one version may not work for a different version, or a more efficient, simpler format might be possible for a different version.

REF: The full details of the export parameters that can be set up for exporting are described in the "FOREIGN FORMAT File Attributes Reference" section.

In many cases, you will be able to use an existing FOREIGN FORMAT file (#.44) entry for your export. If you need to create a *new* FOREIGN FORMAT file entry (rather than using an existing entry), set up the new entry with the Define Foreign File Format option.

Select Fields for Export Option

In the previous step, you determined the data format for your export, and made sure there was a corresponding FOREIGN FORMAT file (#.44) entry. The next step is to choose what file and field data to export. Do this using the Select Fields for Export Option; this creates a SELECTED EXPORT FIELDS template.

The process of creating a SELECTED EXPORT FIELDS template is very similar to the way you choose fields for printing with the Print File Entries option.

REF: For details on selecting fields, see the "Choosing Print Fields" topic in the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

First, you *must* identify the file from which you are exporting data. This is the primary file. Then you choose from which fields to export data.

In addition to fields from that file and its Multiples, you can export data from other files by using the extended pointer syntax.

REF: For more information on pointer syntax, see the "Relational Navigation" section.

Also, you can put other computed expressions at the "EXPORT FIELD:" prompt to make use of Fileman functions or M code.

There are several kinds of specifications that are valid at the "PRINT FIELD:" prompt that are *not* allowed at the "EXPORT FIELD:" prompt. They are:

- WORD-PROCESSING-type fields.
- "**ALL**" signifying all the fields in a file.
- Print qualifiers following the field designation (like "**;X**" or "**;C22**").
- Statistical print qualifiers preceding the field (like "#" or "**&**").
- Backward extended pointers.
- Relational jumps to other files (i.e., use of a terminating colon); instead, use the full extended pointer syntax to obtain data from other files.
- Specifications that return more than one value (e.g., a Multiple in a pointed-to file); you can specify Multiples in the primary file.

After you enter a set of field specifications, you are immediately prompted for a template in which to store the selected fields. You *must* store your field specifications in a template to proceed with the next step in the data export. After you specify a template name for the SELECTED EXPORT FIELDS template, you have completed this step.

The following is an example of the "EXPORT FIELD:" dialogue. The example uses the sample PATIENT file (#200). Several unacceptable responses are shown; the error messages are the ones you would receive to these responses:

```
Select Fileman OPTION: OTHER OPTIONS
Select OTHER OPTION: DATA EXPORT TO FOREIGN FORMAT

Select DATA EXPORT TO FOREIGN FORMAT OPTION: SELECT FIELDS FOR
EXPORT

OUTPUT FROM WHAT FILE: PATIENT

FIRST EXPORT FIELD: NAME;S

SORRY.  You cannot add ;S to the export field specifications.

FIRST EXPORT FIELD: NAME
THEN EXPORT FIELD: INTERNAL(SEX)
THEN EXPORT FIELD: RELIGION:

SORRY.  You cannot jump to another file when selecting fields
for export.

THEN EXPORT FIELD: $E(RELIGION:CLASSIFICATION,1,5)
THEN EXPORT FIELD: DIAGNOSIS <Enter>  (multiple)
  THEN EXPORT DIAGNOSIS SUB-FIELD: DIAGNOSIS
  THEN EXPORT DIAGNOSIS SUB-FIELD: HISTORY <Enter>  (word-
processing)

SORRY.  You cannot choose a word processing field for export.

  THEN EXPORT DIAGNOSIS SUB-FIELD: AGE AT ONSET
  THEN EXPORT DIAGNOSIS SUB-FIELD: <Enter>
THEN EXPORT FIELD: <Enter>
STORE EXPORT LOGIC IN TEMPLATE:  PATIENT TEST
  Are you adding 'PATIENT TEST' as a new PRINT TEMPLATE?  No// Y
<Enter>  (Yes)

Select DATA EXPORT TO FOREIGN FORMAT OPTION:
```

SELECTED EXPORT FIELDS templates are sometimes referred to as PRINT templates in the user dialogue. This is because they are stored in the PRINT TEMPLATE file (#.4).

NOTE: Even though you cannot "jump" to the RELIGION file (#13) using the RELIGION field, which is a pointer to the RELIGION file, you can retrieve data from that file by using extended pointer syntax.

REF: For more information on pointer syntax, see the "Relational Navigation" section.

You can edit a SELECTED EXPORT FIELDS template. The editing *must* occur in the Export Data option, *not* in the standard Print File Entries option. To edit one, enter the template name at the "FIRST EXPORT FIELD:" prompt preceded by a left bracket ("**[**").

If an EXPORT template (see the next step) has been created based on the SELECTED EXPORT FIELDS template that you edit, the SELECTED EXPORT FIELDS template will *not* be updated to reflect the changes. You *must* create a new SELECTED EXPORT FIELDS template to make use of the changes.

Create Export Template Option

The next step to export data is to create an EXPORT template with the Create Export Template option. The EXPORT template combines the SELECTED EXPORT FIELDS template (created in Step 2 above) with a FOREIGN FORMAT (see Step 1 above).

Besides choosing a SELECTED EXPORT FIELDS template and a FOREIGN FORMAT, you will be asked for any additional information that is needed to fully define the export. If you do not supply the requested information, the EXPORT template cannot be created. Values in the FOREIGN FORMAT entry you choose determine whether or not you will be prompted for more information.

The following table indicates which values for which FOREIGN FORMAT fields will result in prompts:

Foreign Format Field	Value	Information Required
FIELD DELIMITER	"ASK"	The character or characters to separate fields.
RECORD DELIMITER	"ASK"	The character or characters to separate records.

Foreign Format Field	Value	Information Required
RECORD LENGTH FIXED?	"1" or "YES"	The number of characters in each field to be exported.
NEED FOREIGN FIELD NAMES?	"1" or "YES"	The name of each field recognized by the importing application.
MAXIMUM OUTPUT LENGTH	"Ø"	The maximum number of characters on each line of output, usually the longest possible exported record.
PROMPT FOR DATA TYPE?	"1" or "YES"	The DATA TYPE field value of each exported field; possible choices are: FREE TEXT NUMERIC DATE/TIME

In the example below, the file and field specifications in the SELECTED EXPORT FIELDS template example (Error: Reference source not found) are combined with the 123 Import Numbers FOREIGN FORMAT:

```
Select DATA EXPORT TO FOREIGN FORMAT OPTION: **CREATE EXPORT
TEMPLATE**

OUTPUT FROM WHAT FILE: **PATIENT**

Enter SELECTED EXPORT FIELDS Template: **PATIENT TEST**
**SELECTED EXPORT FIELDS**  (OCT 30, 1992@11:32) USER #7  FILE
#99002

Do you want to see the fields stored in the PATIENT TEST
template?
Enter Yes or No: NO// **YES**

FIRST PRINT FIELD: NAME// **<Enter>**
THEN PRINT FIELD: INTERNAL(SEX)// **<Enter>**
THEN PRINT FIELD: $E(RELIGION:CLASSIFICATION,1,5)// **<Enter>**
THEN PRINT FIELD: DIAGNOSIS// **<Enter>**
```

```
  THEN PRINT DIAGNOSIS SUB-FIELD: DIAGNOSIS// **<Enter>**
  THEN PRINT DIAGNOSIS SUB-FIELD: AGE AT ONSET// **<Enter>**
  THEN PRINT DIAGNOSIS SUB-FIELD: // **<Enter>**
THEN PRINT FIELD: // **<Enter>**

Do you want to use this template?
Enter Yes or No: YES// **<Enter>**

Do you want to delete the PATIENT TEST template
after the export template is created?
Enter Yes or No: NO// **<Enter>**
```

When asked if you want the SELECTED EXPORT FIELDS template deleted, answer YES only if you know you will not need the template again. If an EXPORT template is not successfully created, the SELECTED EXPORT FIELDS template will not be deleted.

Next, identify the FOREIGN FORMAT to use, and name the EXPORT template that you are creating. You *cannot* overwrite an existing PRINT template:

```
Select FOREIGN FORMAT: **123 IMPORT NUMBERS**

Enter name for EXPORT Template: **PATIENT TO 123**
  Are you adding 'PATIENT TO 123' as
    a new PRINT TEMPLATE (the 197TH)?  No// **Y** **<Enter>**
```

After you choose the EXPORT template name, you are prompted for any additional information needed. In this example, the format does require additional information: the DATA TYPE field value for each field (in this situation the defaults derived by the Export Tool are correct) and the maximum length of each record:

```
Enter the data types of the fields being exported below.

Do you want to continue?
Enter Yes or No: YES// **<Enter>**

NAME: FREE TEXT// **<Enter>**
INTERNAL(SEX): FREE TEXT// **<Enter>**
$E(RELIGION:CLASSIFICATION,1,5): FREE TEXT// **<Enter>**
DIAGNOSIS in DIAGNOSIS subfile: FREE TEXT// **<Enter>**
```

```
AGE AT ONSET in DIAGNOSIS subfile: NUMERIC// <Enter>

Enter the maximum length of a physical record that can be
exported.
Enter '^' to stop the creation of an EXPORT template.

MAXIMUM OUTPUT LENGTH: 100

     Export Template created.
```

The Export Tool checks to make sure that your SELECTED EXPORT FIELDS template does not contain fields from Subfiles (Multiples) that are not descendent from each other.

If you have not followed that restriction, you will receive an error message. The SELECTED EXPORT FIELDS template would have to be modified.

Choose Entries/Export Data

In the final step to export data, use the Export Data option to select which entries from the file to export, and then perform the export.

First, choose which entries to export with a "SEARCH" dialogue; then choose the order of the exported entries with a "SORT BY" dialogue (you are not given the "SORT BY" dialogue, if you are exporting fields from Subfiles.) Finally, specify the device to send the exported data to.

During either the Search or Sort process, you can use previously created SEARCH and SORT templates. Those templates need not have been originally made during a data export; however, SORT templates that contain unacceptable qualifiers should not be used. At the "SORT BY:" prompt, you can only use the following subset of sort qualifiers:

Sort Qualifier	Description
'	To not sort. Used when you want to use the "FROM … TO" dialogue to restrict the entries to be exported.
-	To sort in reverse order.

;Ln	To sort on the first n-characters only.
;TXT	To sort following strict ASCII sorting sequence.

REF: For more detailed information about searching and sorting, see the "Print: How to Print Reports from Files" and "Search" chapters in the *Fileman User Manual*.

Export Example

Here is an example of an export using the "PATIENT TO 123" EXPORT template created in the previous section. You begin by identifying the file and the EXPORT template that you want to use for the export. Do *not* enclose the template's name with brackets. Again, you can delete the EXPORT template after a successful export, if you want.

Because there is a Multiple involved, you are told that you will not have the opportunity to sort. Then, you are given the opportunity to search the file for entries to export.

```
Select DATA EXPORT TO FOREIGN FORMAT OPTION: EXPORT DATA

OUTPUT FROM WHAT FILE: PATIENT// <Enter>

Choose an EXPORT template: PATIENT TO 123 <Enter>


Do you want to delete the PATIENT TO 123 template
after the data export is complete?
Enter Yes or No: NO// <Enter>

Since you are exporting fields from multiples,
a sort will be done automatically.
You will not have the opportunity to sort the data before
export.

Do you want to SEARCH for entries to be exported? NO// YES

  -A- SEARCH FOR PATIENT FIELD: DATE OF BIRTH
  -A- CONDITION: < <Enter> LESS THAN
  -A- LESS THAN DATE: 1980 <Enter>

  -B- SEARCH FOR PATIENT FIELD: <Enter>

IF: A// <Enter>  DATE OF BIRTH LESS THAN 1980

STORE RESULTS OF SEARCH IN TEMPLATE: <Enter>
```

If Multiples had not been involved, you would now be able to respond to the SORT BY dialogue. You can do the same things with sort here that you can do when using the Print File Entries option.

What Device to Send Export Data To

After you complete the sort dialogue, you are immediately given the "DEVICE:" prompt. Choose what device the exported data should be sent to:

```
    DEVICE: <Enter>
```

If you press the **Enter** key at the "DEVICE:" prompt, the export output will be displayed on your *screen*. Sending the formatted export data to the screen allows you to use a PC-based screen capture to put the data into a file. This file would be a readable ASCII file on that computer. This method of transferring the data into a file is a simple one that will often be successful and convenient, especially if the importing application is on the same PC.

When using a screen capture to create a file from the exported data, you *must* consider the peculiarities of your communication and terminal emulation software. Your communication application, for example, may intercept certain control characters (like the **<TAB>**, ASCII 9) and convert them into something else. This may cause the import to fail. Also, your terminal emulation may automatically "break" lines at 80 characters by inserting an unwanted carriage return or line feed. When emulating VT-100 and other ANSI terminals, you can avoid this last problem by turning wraparound mode off.

CAUTION: When exporting data to your terminal's screen, there will be no page breaks. Therefore, there is no graceful way to interrupt the export once it has begun.

Sending Export Data to a Host File

Having data printed on-screen is of little use, if you are using a terminal with no screen-capture capabilities. An alternative is to send the data to a file on the host system, for example, to a VMS file if you are using DSM. Another advantage to sending data to a Host file is that only the exported data will be in the file. (Often, screen captures will unavoidably contain extraneous parts of the user's dialogue prior to or after the export.) To export your data to a file, at the "DEVICE:" prompt, send your export output to an HFS-type device.

Your IRM should be able to help you, if you are not sure how to use HFS devices. The *Kernel Systems Manual* also describes how to send output to Host files, including how to set up and use HFS-type devices.

When a Host file is created, you *must* move that ASCII file to the computer on which the importing application resides. A file transfer protocol, such as KERMIT or XMODEM, can be used to move this file.

The export can be queued, if it is not sent to the screen. Queuing the export is recommended for large files and for complex sorts of the data.

NOTE: On HFS Device Setup on OpenVMS Systems: DSM for OpenVMS requires that you add a command parameter to the OPEN command, if you export records longer than 512 characters to a Host file. The parameter is RECORDSIZE=*nnnn*, where "*nnnn*" is greater than the longest record that you are exporting. If you are using Kernel's DEVICE file (#3.5), the OPEN PARAMETER field for the HFS device you are using should be edited to look like "(NEW:RECORDSIZE=*nnnn*)".

Sample Output

The data below has been prepared for import by Lotus 1-2-3, so it need not be easily read by people. However, you can see that text fields are surrounded by quotes; empty text fields consist just of two quotes (""). A space is in between each field's value. Numeric values have no quotes. If a field defined as numeric in the Fileman data dictionary has no value, a zero (0) is output because this format has SUBSTITUTE FOR NULL set to "0".

```
"FMPATIENT,FIVE" "m" "PROTE" "GANGRENE" 45
"FMPATIENT,SIX" "f" "CATHO" "SLEEPING SICKNESS" 28
"FMPATIENT,SEVEN" "m" "PROTE" "CIRRHOSIS" 25
"FMPATIENT,EIGHT" "f" "OTHER" "FLU" 34
"FMPATIENT,NINE" "m" "" "BLOOD POISONING" 44
"FMPATIENT,FIVE" "m" "PROTE" "GUN SHOT " 50
"FMPATIENT,EIGHT" "f" "OTHER" "FLU" 37
"FMPATIENT,NINE" "m" "" "FLU" 0
"FMPATIENT,EIGHT" "f" "OTHER" "FLU" 46
"FMPATIENT,EIGHT" "f" "OTHER" "APPENDICITIS" 39
```

Special Considerations: Exporting Numbers

If a number comes from a field in your primary file that is defined as NUMERIC or COMPUTED, that number will be exported with all leading spaces or trailing insignificant zeros removed. This is different from the way that the regular Fileman Print File Entries works. If the field had a value of zero, the character zero (0) will be exported. If the value of a numeric field in the primary file is null, the exported value will depend on the contents of the SUBSTITUTE FOR NULL field for the format being used.

If a number comes from a source other than a DATA TYPE field of NUMERIC or COMPUTED in the primary file, it can be output with leading spaces or trailing insignificant zeros. Such a number might originate from a field in a pointed-to file reached by the relational syntax, a Fileman function, or other computed expression. In these cases, the value of the SUBSTITUTE FOR NULL field will usually have no effect on what is exported.

NOTE: Whether exported numbers have leading spaces or trailing insignificant zeros and whether nulls produce special output is controlled by how the field is defined in the Fileman data dictionary. The DATA TYPE field input by the user when the PROMPT FOR DATA TYPE? field contains YES does *not* affect these characteristics of the export.

Special Considerations: Multiples

Data Flattening

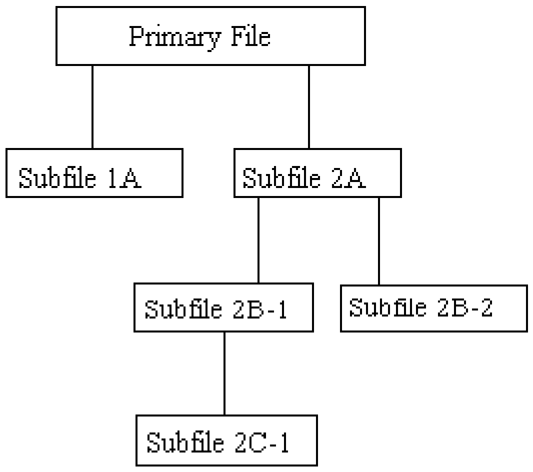
Data exported from Multiples is "flattened;" that is, data at upper levels is repeated for each subentry. For example, take the comma-delimited export for a top level file's #.01 NAME field and a Subfile's #.01 DATE and #1 TYPE fields. The output for an entry with four subentries would look like:

```
    FMPATIENT,01-JAN-95,SC
    FMPATIENT,24-JUN-95,NSC
    FMPATIENT,14-AUG-95,SC
    FMPATIENT,21-JUL-96,NSC
```

NOTE: The top level .01 field is repeated for each Subfile entry.

No More Than One Multiple at Any One File Level

You *cannot* export *more than one* Multiple at *any one file level*. You can export data from one Multiple and from Subfiles directly descendent from that Multiple (as long as you never export more than one Subfile at the same level). Suppose you are exporting data from a file with the following structure:



In addition to fields in the Primary file, you can export from Subfile 1A *or* Subfile 2A, but not from both. Also, you can export from Subfile 2A, Subfile 2B-1, and Subfile 2C-1, but you could not additionally choose fields in Subfile 2B-2. If you need data from Subfiles that are not directly descendent from each other, you can do multiple exports and "join" the data together in the importing application.

Sorting with Multiples

A special, automated sort is done to the data when Multiples are exported; you *cannot* perform your own sort. When Subfiles are involved, the Export Tool performs a special sort in order to format the data. Since the Export Tool *must* do this customized sort, you cannot sort the data yourself. If you need the data in a particular sequence, sort it in the importing application. You can perform any search on the data that is necessary to choose entries for export.

About EXPORT Templates

The Export Tool uses two types of templates: the EXPORT FIELDS template (created in Step 2) and the EXPORT template (created in Step 3). These templates are variations on standard PRINT templates. They are stored in the PRINT TEMPLATE file (#.4) and are sometimes referred to as PRINT templates in the user dialogue. Although similar to PRINT templates, they do differ in important respects. For example, you *cannot compile* either of the Export Tool's templates.

You can delete these templates as soon as they are used if you wish. Also, both kinds of templates can be deleted using the Template Edit option on the Utility Functions menu [DIUTILITY]. In addition, you can delete an EXPORT FIELDS template by choosing the template within the Select Fields for Export option, editing it, and putting an at-sign ("**@**") at the "NAME:" prompt. Do *not* delete an EXPORT template before a queued export has been completed.

How to Import Data

The following menu shows the option used to import data:

```
    Fileman …   [DIUSER]
      Other Options ...   [DIOTHER]
          Import Data   [DDMP IMPORT]
```

The Import Tool lets you import records stored in an ASCII data file into a Fileman file.

The Import Tool imports records from an ASCII data file by *adding* them as new records to the Fileman file in question. Existing records in the destination Fileman file are never edited or updated, and the Import Tool does not prevent duplicate records from being added.

Importing data records from an ASCII file is a four-step process, as described below.

Generate ASCII Source File

Generate your source file (from your non-Fileman application), containing the records to be imported. Generate the file with one record per line, with the fields in each record being set off using either the delimited or fixed-length method. The last record in the file *must* be terminated with the appropriate EOL (end-of-line) character(s) for your operating system.

Once you generate your ASCII source file, you need to move it to a disk that is accessible from the computer system running Fileman. Your IRM should be able to assist you with this.

Specify Data Format, Source File, and Destination File

Invoke Fileman's Import Data option. It loads a two-page Screenman form. On page one of the form, you need to specify the: data format, source file, and destination file for your import.

DATA FORMAT—INTERNAL or EXTERNAL: Specify if the incoming data is in external form (the way Fileman would display it) or internal form (the way Fileman would store it). Unless you are knowledgeable about how Fileman stores data, you should choose EXTERNAL. Also, the incoming data is only validated by Fileman if you choose EXTERNAL (validation prevents you from putting invalid data into the file).

FOREIGN FORMAT: Choose a Foreign Format entry whose settings match the ASCII format for the incoming records. The only settings used from the Foreign Format entry are Record Delimiter, Record Length Fixed?, and Quote Non-numeric Fields?. Make sure the settings in the Foreign Format match the format of your incoming data. Because some foreign applications export data in a different format than they import it, a Foreign Format that works for export may not have the appropriate settings for import.

As an alternative to specifying a Foreign Format entry, you can manually specify the settings for your incoming data in the three provided fields:

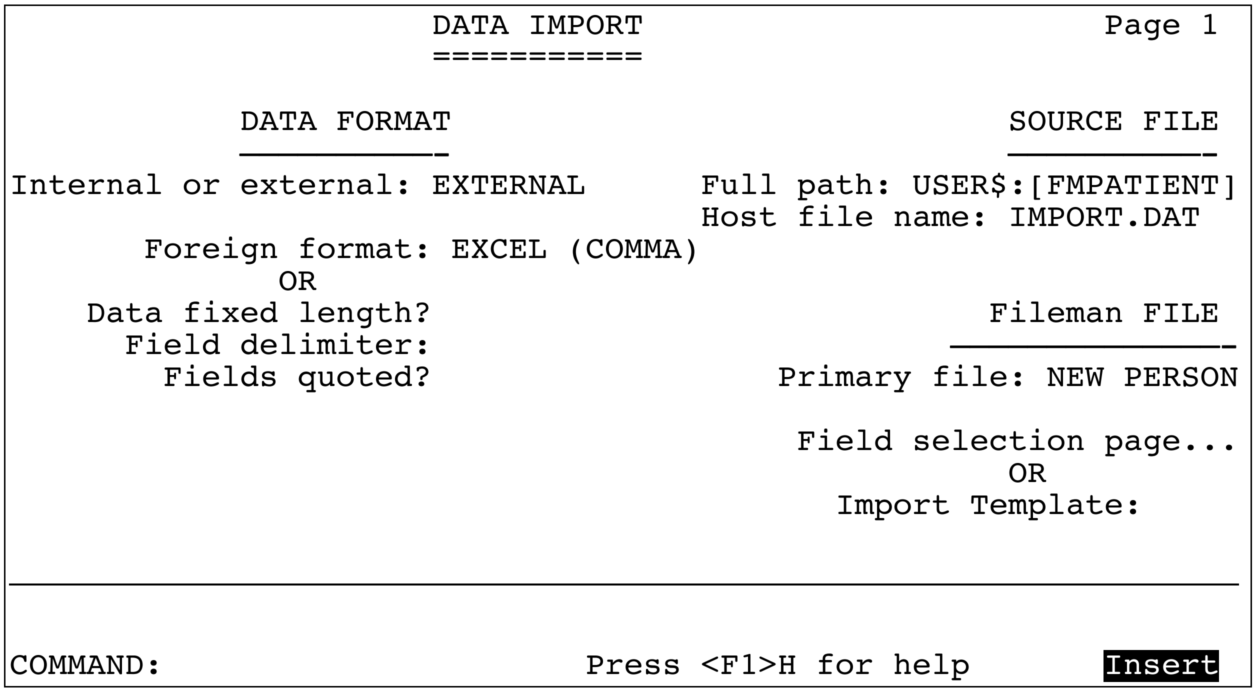
1. Is the data fixed length?
2. If not, what is the field delimiter?
3. Are fields quoted?

SOURCE FILE: Enter the path and name of your source file (the file containing the records to import).

Fileman FILE: Specify the destination file for the imported records.

FIELD SELECTION PAGE/IMPORT TEMPLATE: This is where you match the fields in the incoming records to the fields in the destination file. If you do not have an existing IMPORT template that matches incoming to destination fields, go to the Field Selection page and specify those fields individually (see the "Match Source to Destination Fields" section).

A completed page one of the form might look like:



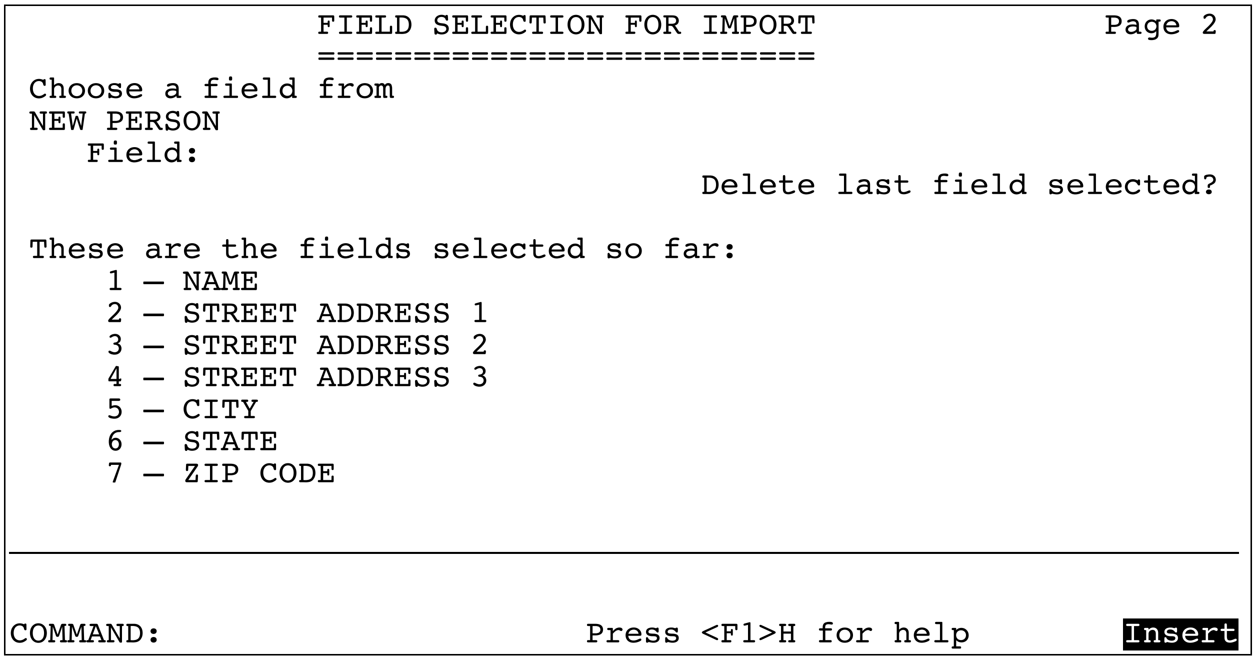
```
                      DATA IMPORT                        Page 1
                      ===========

            DATA FORMAT                             SOURCE FILE
            -----------                             -----------
Internal or external: EXTERNAL      Full path: USER$:[FMPATIENT]
                                    Host file name: IMPORT.DAT
       Foreign format: EXCEL (COMMA)
             OR
    Data fixed length?                             Fileman FILE
      Field delimiter:                             ------------
        Fields quoted?                  Primary file: NEW PERSON

                                         Field selection page...
                                                    OR
                                           Import Template:

----------------------------------------------------------------

COMMAND:                      Press <F1>H for help        Insert
```

Match Source to Destination Fields

For your import, you need to match each field in the incoming record to a field in the destination Fileman file.

Fields in the incoming record are imported in order, from left to right.

Thus, for each field in the incoming record, you specify the corresponding destination field in the Fileman file, in the same order. The first Fileman field you specify will be the destination for the first field in the incoming record, the second will match the second field in the incoming record, and so forth.



```
                FIELD SELECTION FOR IMPORT               Page 2
                ==========================
 Choose a field from
 NEW PERSON
    Field:
                                         Delete last field selected?

 These are the fields selected so far:
     1 — NAME
     2 — STREET ADDRESS 1
     3 — STREET ADDRESS 2
     4 — STREET ADDRESS 3
     5 — CITY
     6 — STATE
     7 — ZIP CODE


COMMAND:                     Press <F1>H for help        Insert
```

Remember that you *must* include the .01 field, and any fields that are required identifiers for the top level of the file. The same is true for any Subfiles (Multiples).

If you specified a fixed-length (as opposed to delimited) data format for the incoming records, you *must* enter not only the destination Fileman field, but also the length for each corresponding incoming field.

Each time you enter a field at the "Field:" prompt, it's added to the bottom of the list of fields displayed on the form. This shows you the destination fields you have selected, and their order. If you make a mistake, you can delete fields from the bottom of the list, one-by-one, by entering YES at the "Delete last field selected?" prompt. To insert a field, delete back to the insertion point, enter the new field, and then re-enter the deleted fields.

REF: There are special issues when importing data into fields in Multiples; see the "Special Considerations: Multiples" section.

You can save the information you specify on the Field Selection page in an IMPORT template. This lets you reuse the field matching criteria you have entered for subsequent imports that use the same file and fields, without having to re-enter it. To save your field specifications as an IMPORT template, answer YES to the "Do you want to store the selected fields in an Import Template?" question, which you are asked after you exit the Import form (see Run the Import below). Then, for future imports, simply enter the name of the IMPORT template on Page 1 of the Import form. You can use any IMPORT template to which your Fileman Access Code gives you access.

Run the Import

Once you have set up your data format, source file, and destination file, and matched source to destination fields, exit the Import form (press **<F1>E**). After you exit the form, you are asked a series of questions:

1. Do you want to store the selected fields in an Import Template?
2. Do you want to proceed with the import?
3. Device for Import Results Report

Storing your file and field specifications in an IMPORT template lets you do subsequent imports without having to re-enter all of the field information.

If you proceed with the import, enter a device to which the Import Results report should print. You can run the Import directly or queue it.

As the import proceeds, if an error occurs updating a field in a particular record, the record will not be added, and an error message is added to the Import Report saying what the problem was.

An example of the dialogue after exiting the Import form is shown below:

```
    Do you want to store the selected fields in an Import
Template? **YES**

    Name of Import Template: **ZZIMPORT**
      Are you adding 'ZZIMPORT' as a new Import Template? **YES**

    Do you want to proceed with the import? **YES**

    Device for Import Results Report: HOME// **<Enter>**
```

Once the import finishes, you can review the Import Results report. It lists:

- The criteria you chose for your import.
- Any records for which the import failed.
- The internal entry numbers of the first and last records imported.

Here's a sample Import Results report:

```
                Log for Fileman Data Import             Page 1
               ==============================

             Import Initiated By: 10 FMPATIENT

              Source File: USER$:[FMPATIENT1]IMPORT.DAT
                    Fixed Length: NO
                    Delimited By: ,
              Text Values Quoted: NO
                      Values Are: External

        Primary FileMan Destination File: NEW PERSON

Seq  Len  Field Name                Subfile Name (if applicable)
---  ---  ----------                ----------------------------
1    n/a  NAME
2    n/a  STREET ADDRESS 1
3    n/a  STREET ADDRESS 2
4    n/a  STREET ADDRESS 3
5    n/a  CITY
6    n/a  STATE
7    n/a  ZIP CODE


                          Error Report
                          ------------
Record #4 Rejected:
   The value 'Illlinois' for field STATE in file NEW PERSON is
not valid.


                       Summary of Import
                       -----------------

             Total Records Read: 7
            Total Records Filed: 6
         Total Records Rejected: 1

      IEN of First Record Filed: 209
       IEN of Last Record Filed: 214

        Import Filing Started: Jul 16, 1996@08:24:36
      Import Filing Completed: Jul 16, 1996@08:24:38
        Time of Import Filing: 0:00:02
```

In this example, six records were added, and one record was not added. The record that was not added was the fourth record in the source file. It failed due to the misspelled value "Illlinois" being rejected by the STATE field in the NEW PERSON file (#200).

Importing Data Into Multiples

CAUTION: Incoming Data Should *not* be Flattened.

The Import Tool expects that any data bound for a Multiple be contained in the same import record (line of data) as the data for the top file level. This is different from the output of the Export Tool, which "flattens" exported data from Multiples into separate lines of output.

For example, consider a comma-delimited import of records, each including a name plus four subentries. Each subentry contains a DATE and a TYPE. The records will be imported into a file with a top-level NAME field (#.01), and a Multiple with DATE field (#.01) and TYPE field (#1). For this import, you would choose the destination fields as follows:

```
                   FIELD SELECTION FOR IMPORT            Page 2
                   ==========================
 Choose a field from
 PATIENT : DATE    Subfile
    Field:
                                             Delete last field selected?

 These are the fields selected so far:
     1 — NAME
     2 — DATE:DATE
     3 — DATE:TYPE
     4 — DATE:DATE
     5 — DATE:TYPE
     6 — DATE:DATE
     7 — DATE:TYPE
     8 — DATE:DATE
     9 — DATE:TYPE

_____________________________________________________________________
 Exit    Save      Next Page     Refresh

 Enter a command or '^' followed by a caption to jump to a
specific field.


 COMMAND: NEXT                      Press <F1>H for help       Insert
```

A corresponding line of data to be imported for a record, containing data for both the top-level record and its subentries, would look like:

```
FMPATIENT,01-JAN-95,SC,24-JUN-95,NSC,14-AUG-95,SC,21-JUL-96,NSC
```

NOTE: You *must* file the same number of subentries in each record you import.

Completeness of Subfile Entries

New subentries need to be added to every Subfile on a path to the lowest level Subfiles. Your data *must* include values for the .01 field and all the required identifiers for every Subfile (as well as for the top level of the file). You can add more than one subentry in a particular Subfile. However, you are restricted to the same set of fields for every entry in each Subfile.

Importing from VMS Files

When importing from a data file that's been transferred to a VMS-based computer system, a problem can occur if, once transferred, the data file does not get a maximum record length stored in its file header. This can happen when a DOS file is moved to a VMS system by some protocols. When the maximum record length is unknown, VMS uses a default maximum size of 510. If the length of a data record in the source file is larger than the maximum size, an error results.

The solution is to run the VMS CONVERT utility on the Host file. This utility adds the maximum record information to the file header and everything will work just fine!

You can see if the maximum record length is stored in a file's header on a VMS system, by using the following DCL command:

```
    DIR filename /FULL
```

Foreign Formats

FOREIGN FORMAT File Attributes Reference

The following fields in the FOREIGN FORMAT File (#.44) correspond to attributes of the formatted data that you wish to export or import:

- FIELD DELIMITER
- QUOTE NON-NUMERIC FIELDS?
- SEND LAST FIELD DELIMITER?
- PROMPT FOR DATA TYPE?
- RECORD DELIMITER
- SUBSTITUTE FOR NULL
- RECORD LENGTH FIXED?
- DATE FORMAT
- MAXIMUM OUTPUT LENGTH

- FILE HEADER
- NEED FOREIGN FIELD NAMES?
- FILE TRAILER

When *exporting* records, all fields in this file are used in the export process. When *importing* records, only three fields are used in the import process:

- FIELD DELIMITER
- RECORD LENGTH FIXED?
- QUOTE NON-NUMERIC FIELDS?

In this section, each format characteristic is described. Some combinations of characteristics are unacceptable; these situations are mentioned.

Also, some of the fields allow you to enter M code. Export-specific variables you can use in this M code are described in the section Variables Available for Programmer Use.

To set up a FOREIGN FORMAT file (#.44) entry, use the Define Foreign File Format option to print out a format, use the Print Format Documentation option.

FIELD DELIMITER

Many applications can import and export data, if the values of fields in each record are separated by a known character or sequence of characters. The application puts (or expects) data before the first delimiter into its first field, between the first and second delimiter into the second field, and so on. Therefore, the ability to specify and recognize these field delimiters is a crucial aspect of many data exchanges.

The Import and Export Tools' FIELD DELIMITER fields allow you to specify up to 15 characters to be placed between each field. You can directly enter any string of characters except ones that begin with a number or consist of characters that have special meaning when editing Fileman data (e.g., "**^**" or "**@**").

If your field delimiter begins with one of these restricted characters or consists of an unprintable control character (like **<TAB>**), you can enter the ASCII-value of the delimiter. When entering the ASCII values, always use three digits. Thus, **<TAB>** (ASCII 9) becomes "009" and "**@**" (ASCII 64) becomes "064". You can enter up to four ASCII values. If more than one is needed, separate the values with commas (e.g., "048,094").

If you want the user to be prompted for a field delimiter at the time the EXPORT template is being created, enter "ASK" in this field.

CAUTION: Using unprintable control characters (ASCII values less than 32) as delimiters may not have the effect you want. During either export or import, often control characters are intercepted by terminal software, communication programs, or network links; they may not be passed through unaltered as regular printable characters usually are. For example, ASCII value 5 is interpreted by many terminals as a request for their Answerback Message. Thus, putting "005" in the FIELD DELIMITER field might cause an Answerback Message to be returned by your terminal instead of the ASCII value 5 being inserted between field values.

NOTE: The importing application will find the delimiting character, if it occurs in the data. This will cause an incorrect determination of the boundary between fields. For example, if a comma (**,**) is the field delimiter and the data for a field was **FMPATIENT,10**, the importing application would put **FMPATIENT** into the first field and **10** into the second field. You can avoid this problem by specifying that data in non-numeric fields be surrounded by quotes (e.g., **"FMPATIENT,10"**). Most importing applications will ignore delimiters, if they occur within a quoted string. See the discussion of Quote Non-numeric Fields? below.

SEND LAST FIELD DELIMITER?

Some importing applications expect a field delimiter following every field, including the final field in a record. Other applications only expect delimiters between fields; nothing follows the final field. This field allows you to specify whether or not a field delimiter should be exported after the last field. A YES answer sends the delimiter, a NO answer does not.

The contents of this field does *not* affect whether or not a delimiter is sent after each *record*.

RECORD DELIMITER

Applications that import delimited fields need to know when one record ends and a new one begins. In most cases, records are separated by a carriage return (or by a line feed and a carriage return). This is the same as pressing the **Enter** key at the end of a line. The Export Tool *automatically* puts this separator after each record; every record begins on a new line of output. You do *not* need to put the ASCII values for carriage return and line feed in this field.

Some applications may also require that additional characters be placed after each record. If this is the case, put those characters into the RECORD DELIMITER field. The requirements for coding the field are the same as for the FIELD DELIMITER field.

RECORD LENGTH FIXED?

A second common way to import and export data (in addition to using delimited data) is with fixed length records. In a fixed length record, each field has a predetermined, constant data length. For example, a name field might be 30 characters long. The name "FMPATNT,10" is only 10 characters long; thus, 20 spaces would be added to the field value to fill the required 30 characters. The next field's value would begin in the thirty-first column.

If you want to import or export fixed length records, answer YES to this field. At the time that the EXPORT template is created (or an import is done), the user will be prompted for the length of each field in the target or source file.

During export, in most cases data will be truncated when the length of a field is reached. Thus, if a field contains 32 characters but the user-defined length is 30, the last 2 characters will not be exported. However, DATE/TIME-valued fields will always be exported in their entirety. For dates, the user *must* indicate a data length at least as long as the exported date, which is 11 characters for standard Fileman dates.

NOTE: Fixed record lengths cannot be used in conjunction with field delimited data. Also, the *maximum record size for exports for a fixed length format is 255* characters. There is *no limit on record length during import*, however.

CAUTION: Fixed length exports will succeed only if all fields are exported on the same physical line. Therefore, the total of all the field lengths must not be more than the value stored in the MAXIMUM OUTPUT LENGTH field.

MAXIMUM OUTPUT LENGTH

In many cases, data import will be much easier if an entire record is contained on a single "line" of output. That is, there are no carriage returns within a single record, only between records. (This is a requirement for a successful fixed length export.)

In a regular Fileman print, the amount of data printed before a carriage return is dependent on the type of device being used for output—a CRT screen would normally have 80 characters on a line, a printer 80 or 132. For data export, however, the physical characteristics of the output device is not controlling. Rather, the capabilities of the application importing data is overriding. Therefore, you can use the MAXIMUM OUTPUT LENGTH field to specify how long a physical record will be. For field delimited (as opposed to fixed length) exports, this record length can be larger than the traditional M data limit of 255 characters.

Put a number from 0 through 9999 into this field. The default record length is 80. If you want the user to be prompted for a record length at the time that an EXPORT template is being created, put "0" (zero) into this field.

Regardless of the length of the maximum record, a carriage return will be written after each record is output.

NOTE: The length of a record cannot exceed 255 characters when using a fixed length format.

CAUTION: When sending exports to a Host file on a DSM for OpenVMS (e.g., VAX) system, you must add a parameter to the OPEN command, if any of your exported records are longer than 512 characters. See the "Export Data" section for details.

NEED FOREIGN FIELD NAMES?

If this field is answered YES, the user is prompted for a field name for each exported field when the EXPORT template is created. The field names are stored in the NAME OF FOREIGN FIELD field in the EXPORT FIELD Multiple in the PRINT TEMPLATE file (#.4).

REF: For one way to use this information, see the discussion in the "FILE HEADER" section.

QUOTE NON-NUMERIC FIELDS?

When ***importing*** data, Fileman will ignore the field delimiter in a quoted string when this field is set to YES.

When ***exporting*** data, if you want all values that do not belong to a DATA TYPE field of NUMERIC to be surrounded by quotation marks, answer YES to this field.

Many importing applications treat data within quotation marks (") in a special way. Sometimes such data is automatically considered to be text, as opposed to numbers. Also, the importer may ignore the field delimiter character, if it falls within a quoted string. Quoting a null value from a non-numeric field will result in two double quotes ("") being exported.

During export, the DATA TYPE field value is automatically determined for fields in the primary file and its Multiples. DATA TYPE fields of NUMERIC are considered NUMERIC. There may be other fields that you want treated as NUMERIC. For example:

- COMPUTED-type fields with numeric results.
- Fields referenced by the extended pointer syntax.
- Replies to the "EXPORT FIELD:" prompt that are computed expressions with numeric results.

By default, these fields are assigned a FREE TEXT DATA TYPE. If you want the user to choose the DATA TYPE when the EXPORT template is created, answer YES to the PROMPT FOR DATA TYPE? field.

If the Export Tool assigns a non-numeric to a DATA TYPE field or if the user chooses one of those DATA TYPE field values, the field's values will be surrounded by quotes when this field contains YES.

NOTE: Do *not* set this field to YES if a fixed length record is being exported or imported.

PROMPT FOR DATA TYPE?

The Export Tool will determine the DATA TYPE field value for fields in the primary file and its Multiples based on their definition in the data dictionary. Other fields are automatically assigned a DATA TYPE of FREE TEXT. If you want the user to choose the DATA TYPE of each field when creating an EXPORT template, answer YES to this field. The only DATA TYPE field values recognized by the Export Tool are the following:

FREE TEXT
NUMERIC
DATE/TIME

The DATA TYPE field value entered by the user controls whether or not the values from that field will be surrounded by quotes if the QUOTE NON-

NUMERIC FIELDS? field is set to YES. The user supplied DATA TYPE field value does *not* affect how numbers are exported; numeric export is controlled by the DATA TYPE field value in the data dictionary only.

SUBSTITUTE FOR NULL

Numeric fields with no data (a "null" value) will result by default in nothing being exported for that field. For fixed record length exports, this should not be a problem. However, if your importing application uses spaces as a delimiter, you may need a printable character to be exported for null-valued numeric fields. If you want a character or characters (such as "**0**" or "**.**") substituted for numeric nulls, put them into this field. Null values for DATA TYPE field values of NUMERIC in the primary file (including its Multiples) will have this character exported. If you want quotes (**"**) in your substitute string, enter two quote marks (**""**) for each quote you want.

NOTE: Do *not* put anything in this field when defining a fixed length format.

CAUTION: There will be no substitution for null values if the field being exported is not in the primary file; that is, if it is reached using relational navigation.

DATE FORMAT

The native, or default, format for dates varies from application to application. Fileman uses two formats:

- Internal or Storage format: *YYYMMDD* Where *YYY* is the year minus 1700.
- External or Default display format: *MON DD,YYYY*

When data from a DATA TYPE field of DATE/TIME is exported, it is in the external format.

Since the importing application may recognize a different format, you can change the exported value by placing M code in this field (only those with programmer access can enter code in this field.) When this M code is executed, the local variable X will contain the date in Fileman *internal* format. Your M code should result in the local variable Y containing the date in the format you want exported.

If your format will be used with Kernel, it is recommended that you make use of the date extrinsic functions provided by Kernel, if possible.

REF: For more information on Kernel date extrinsic functions, see the *Kernel Developer's Guide*.

Data from fields with DATA TYPE field values of DATE/TIME in the primary file, its Multiples and pointed-to files are altered by the code in this field; date values from other sources are not. There is another way to change the exported output; you can use a Fileman function when selecting fields for export:

```
  THEN EXPORT FIELD: NUMDATE(DATE OF BIRTH)
```

The DATE FORMAT field will have no effect on that output.

FILE HEADER

Some applications require special information to process the data in the file that is imported. For example, the field names might be needed. Also, you may want to put some special data into the file for identification or documentation purposes.

The FILE HEADER field allows you to output information before the stream of exported data. This field can contain either a literal string surrounded by quotation marks (e.g., "Data for Lotus 1-2-3") or M code that, when executed, will write the desired output.

You can put M code here only if you have programmer access. The local variable DDXPXTNO, which equals the internal entry number in the

PRINT TEMPLATE file (#.4) of the EXPORT template being used for data output, is defined when the code is executed. You can use this variable to access information about the export. The data type, length, and foreign field name are stored in the EXPORT FIELD Multiple (#100).

REF: For additional information, see the data dictionary for the PRINT TEMPLATE file.

FILE TRAILER

You can use this field like the FILE HEADER field. The literal or M code will be output after the exported data.

Variables Available for Programmer Use

Some of the fields in the FOREIGN FORMAT file allow you to enter M code, if you have programmer access. You may want to use data stored in the EXPORT template entry at the time the export is performed. You may also want to access information in the FOREIGN FORMAT file entry used for the export.

Two variables are available for use in the M code entered in FOREIGN FORMAT file fields:

DDXPXTNO—The internal entry number of the EXPORT template in the PRINT TEMPLATE file.
DDXPFFNO—The internal entry number of the Foreign Format in the FOREIGN FORMAT file.

Consult the data dictionaries of the two files for fields that may contain useful information about either the format or the specific export itself. The EXPORT FIELD Multiple in the PRINT TEMPLATE file might be of particular interest. This Multiple contains information about each field being exported.

Print Format Documentation Option

```
Fileman ...                           [DIUSER]
  Other Options ...                   [DIOTHER]
    Data Export to Foreign Format ...[DDXP EXPORT MENU]
      Print Format Documentation     [DDXP FORMAT DOCUMENTATION]
```

You can list the available FOREIGN FORMAT file entries on the system using the Print Format Documentation option. When you use this option, you are given the choice of specifying individual formats or of printing all of the formats on your system. Since your system can contain many formats, try to select individual ones.

A typical dialogue for choosing formats and the resulting output looks like this:

```
Select DATA EXPORT TO FOREIGN FORMAT OPTION: PRINT FORMAT
DOCUMENTATION

     Select one of the following:
          1         Only print selected foreign formats
          2         Print all foreign formats

Enter response: 1 <Enter>  Only print selected foreign formats

Select FOREIGN FORMAT: 123 IMPORT NUMBERS

Select FOREIGN FORMAT: EXCEL-COMMA

Select FOREIGN FORMAT: <Enter>

DEVICE: <Enter>


AVAILABLE FOREIGN FORMATS            NOV  2,1992 15:34    Page 1
-----------------------------------------------------------------
NAME: 123 IMPORT NUMBERS

DESCRIPTION:  This format exports data for use with LOTUS 1-2-3
spreadsheets. Non-numeric fields will be in quotes. Each field
will be separated by a space.

USAGE NOTE:  To import into 1-2-3, choose FILE->IMPORT->NUMBERS.
```

```
OTHER NAME:  LOTUS 123 (NUMBERS)

DESCRIPTION:

NAME: EXCEL-COMMA

DESCRIPTION:  Use this format to export data to the EXCEL
spreadsheet on the Macintosh. The exported data will have a
comma between each field's value. The user will be asked to
specify the data type of each exported field.  Those fields that
are not numeric will be surrounded by quotes ("). Commas are
allowed in the non-numeric data, but quotes (") are not.

USAGE NOTE:

OTHER NAME:  COMMA DELIMITED

DESCRIPTION:  Exported data is delimited by commas.  Non-numeric
data is surrounded by quotes.

OTHER NAME:  CSV

DESCRIPTION:  Comma Separated Values.
```

Define Foreign File Format Option

```
Fileman ...                                     [DIUSER]
  Other Options ...                             [DIOTHER]
    Data Export to Foreign Format ...           [DDXP EXPORT MENU]
      Define Foreign File Format                [DDXP DEFINE FORMAT]
            **> Locked with DDXP-DEFINE
```

All exports depend on a Foreign Format. In addition, you can use Foreign Formats for imports as well. Usually, you will be able to use an existing format to properly format your data for export or import.

To find out what formats exist on your system, see the Print Format Documentation section. If no existing format meets your needs, use the Define Foreign File Format option to create a new one. You can use the Define Foreign File Format option to:

- Define a new Foreign Format from scratch.

- Modify a Foreign Format that has *not* been used to create an EXPORT template.
- Copy an existing format in order to create a similar, modified one.

If you are using the Export Tool through Kernel's menu system, you need the DDXP-DEFINE key to use the Define Foreign File Format option.

The following is an example of making a new format from an existing one:

The Define Foreign File Format option is the first one on the Data Export to Foreign Format submenu:

```
Select OPTION: OTHER OPTIONS
Select OTHER OPTION: DATA EXPORT TO FOREIGN FORMAT

Select DATA EXPORT TO FOREIGN FORMAT OPTION: DEFINE FOREIGN FILE
FORMAT
```

You are first asked for the name of a format. If you want to create a new format from scratch, enter a new name. You will be presented with the Screenman form used to define a Foreign Format.

NOTE: Whenever you are asked to choose a FOREIGN FORMAT, you can reply with either the format's NAME or one of its OTHER NAMES.

Here, an existing format's name is given:

```
Select FOREIGN FORMAT: 123 IMPORT NUMBERS

123 IMPORT NUMBERS foreign format has been used to create an
Export Template.
Therefore, its definition cannot be changed.
```

This format has already been used to create an EXPORT template. Since that template relies on the information in the FOREIGN FORMAT file's entry at the time the template was created, you *cannot* modify this format. Instead, you are given the option of seeing what is in the format:

```
Do you want to see the contents of 123 IMPORT NUMBERS format?
NO// YES

NAME: 123 IMPORT NUMBERS                FIELD DELIMITER: 032
  MAXIMUM OUTPUT LENGTH: 0              FORMAT USED?: YES
  QUOTE NON-NUMERIC FIELDS?: YES        PROMPT FOR DATA TYPE?:
YES
  SEND LAST FIELD DELIMITER?: YES       SUBSTITUTE FOR NULL: 0
 DESCRIPTION:  This format exports data for use with LOTUS 1-2-3
 spreadsheets.  Non-numeric fields will be in quotes.  Each
field
 will be separated by a space.  A 0 will be exported for null-
 valued numeric fields in the primary file.
 USAGE NOTES:  To import into 1-2-3, choose FILE->IMPORT-
>NUMBERS.
```

As this example shows, the FORMAT USED? field is YES. This indicates that the format has been used to create an EXPORT template.

Whether you ask to see the contents of the format or not, you are next given the chance to make a copy of the format in order to modify it. You enter a name for the new format that does not yet exist in the FOREIGN FORMAT file:

```
Do you want to use 123 IMPORT NUMBERS as the basis
for a new format?  NO// YES <Enter>

Name for new FOREIGN FORMAT: CLONE 123 IMPORT NUMBERS

  Are you adding 'CLONE 123 IMPORT NUMBERS' as
    a new FOREIGN FORMAT (the 22ND)?  No// Y <Enter>
```

When the new format has been created, you are given the opportunity to modify it. The Screenman form that follows is used for editing Foreign Formats:

```
          FOREIGN FILE FORMAT: CLONE 123 IMPORT NUMBERS   Page 1
          =============================================

     FIELD DELIMITER: 032             RECORD LENGTH FIXED?
  SEND LAST DELIMITER? YES          MAXIMUM OUTPUT LENGTH:   0
     RECORD DELIMITER:            NEED FOREIGN FIELD NAMES?
```

```
          FILE HEADER:
         FILE TRAILER:
          DATE FORMAT:

  SUBSTITUTE FOR NULL: 0
    QUOTE NON-NUMERIC? YES
 PROMPT FOR DATA TYPE? YES


                             Go to next page to document format.
________________________________________________________________


COMMAND:                      Press <F1>H for help      Insert
```

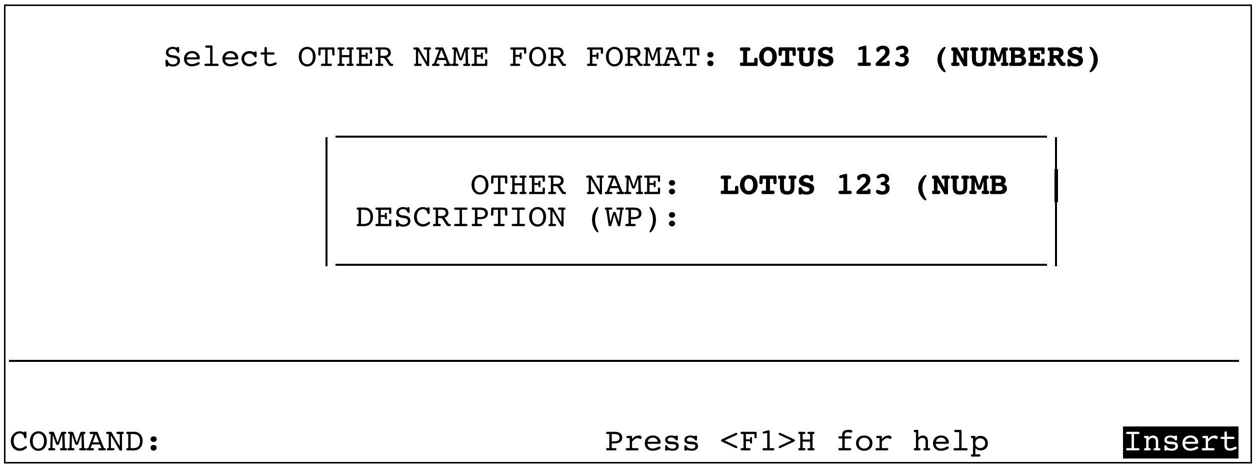
The meaning of the fields on this page of the form is described in the FOREIGN FORMAT File Attributes Reference section. You are presented with the same form whether you are modifying an existing format or creating one from scratch.

TIP: It is important to always create and edit formats using the Data Export options because validity checks on the relationships between the various fields are built into the Screenman form. If you enter inconsistent data, you will be alerted when you try to exit the form.

There is a second page of the form that contains documenting information about the format. The second page allows you to enter a description and usage notes for the format. You can also enter other names for the format (in a Multiple); these other names can then be used to reference the format anywhere in the Export or Import Tools.

Here is what the second page looks like with the Multiple's "popup" window opened:

```
         FOREIGN FILE FORMAT: CLONE 123 IMPORT NUMBERS    Page 2
         =============================================

                    DESCRIPTION (WP):

                    USAGE NOTES (WP):
```



```
        Select OTHER NAME FOR FORMAT: LOTUS 123 (NUMBERS)

                  ____________________________________
                 |                                    |
                 |     OTHER NAME:  LOTUS 123 (NUMB   |
                 | DESCRIPTION (WP):                  |
                 |____________________________________|



_______________________________________________________________

COMMAND:                       Press <F1>H for help       Insert
```

After you have completed and filed the Screenman forms, you are returned to the Data Export submenu. You can now use the new format to create an EXPORT template or do an import.

Chapter 11: Transferring File Entries

The Transfer Entries menu contains three options:

Transfer File Entries
Compare/Merge File Entries
Namespace Compare

These options allow you to transfer entries from one file to another, to compare and/or merge two entries in a single file, or to compare file data dictionaries and entries between two UCIs.

For example, you may need to combine data from two different entries into one of the two. This could happen in a patient database when the same patient has been inadvertently entered twice with the name spelled slightly differently.

⚠ CAUTION: Once you have merged file entries, the merging cannot be undone. Care must be taken that data is not mistakenly lost.

Transfer File Entries Option

The Transfer File Entries option can be used for several purposes. You can use it to:

- Merge two entries in the same file
- Transfer one or more records from one file to another file
- Copy a data dictionary into a new file

However, the Compare/Merge File Entries option (described in Section) should usually be used to merge entries in the same file; it is specifically designed for that task.

You *must* have READ access for the file you are transferring from and

WRITE access for the file you are transferring to. If you are deleting entries after the transfer, you need delete access as well.

Transferring Data Within the Same File

You can use the Transfer File Entries option to merge two entries that are in the same file. To do this:

1. Identify the input and output file as the same
2. Identify the two entries

Data values are then transferred from the FROM entry to the TO entry. The following example shows the simple dialogue:

```
Select OPTION: TRANSFER ENTRIES

Select TRANSFER OPTION: ?
 ANSWER WITH TRANSFER OPTION NUMBER, OR NAME
CHOOSE FROM:
   1           TRANSFER FILE ENTRIES
   2           COMPARE/MERGE FILE ENTRIES

Select TRANSFER OPTION: 1 <Enter>  TRANSFER FILE ENTRIES

INPUT TO WHAT FILE: PATIENT
TRANSFER FROM WHAT FILE: PATIENT
TRANSFER DATA INTO WHICH PATIENT: RECIPIENT,ROGER
TRANSFER DATA FROM PATIENT: FMUSER,FIVE
WANT TO DELETE THIS ENTRY AFTER IT'S BEEN TRANSFERRED?  NO//
```

If the TO entry (Roger Recipient in our example) already has a value on file for a given field, that value will be preserved (i.e., it will *not* be overwritten by the corresponding field value in the FROM entry). This rule applies to word processing data fields also. If the recipient has any text on file, corresponding text from the sender's entry is not merged with it. Thus, if you decide to delete the FROM entry (Sam Sender here), you may lose some data.

In the case of distinct Multiple-valued subfields, merging will take place. The subentries in the FROM entry would be added to the Multiple in the

TO entry. Further, if two subentries have the same .01 value and if any of the subfields are blank in the TO entry, the FROM subentry's data will be placed in the blank subfield. In this way, data is added to Multiples in the same way that it is added to files.

For example, suppose DIAGNOSIS is the label of the .01 field of a Multiple and AGE AT ONSET is a subfield in that Multiple. If Roger Recipient has a DIAGNOSIS of "Angina" and Sam Sender has one of "Diabetes", Recipient ends up with both "Angina" and "Diabetes". Further, if both Recipient and Sender had "Angina", but Recipient had no AGE AT ONSET for that subentry and Sender did have one, the Sender's AGE AT ONSET data for "Angina" would be transferred to Recipient.

The following example illustrates the transfer of data values from one entry to another in more detail. (These two entries will also be used in the discussion of the Compare/Merge File Entries option.) Before the transfer, the Inquire to File Entries option displays these two entries from the SCHOLAR file:

```
NAME:  FMPATIENT,23 A.  SSN:  000-99-9999
    SUBJECT AREA:  PHILOSOPHY
 TOPICS:  23'S PARADOX
 TOPICS:  PRINCIPLE OF EXCLUDED MIDDLE
 TOPICS:  SET THEORY
 TOPICS:  THEORY OF TYPES
 TOPICS:  LIAR PARADOX

 NAME:  FMPATIENT,23 H.  DATE OF BIRTH:   1872   SSN:  000-88-
8888
    SUBJECT AREA:  MATHEMATICS
  TOPICS:  24'S PARADOX
  TOPICS:  SET THEORY
  TOPICS:  THEORY OF TYPES
  TOPICS:  AXIOM OF INFINITY
```

The transfer is then initiated with the following dialogue:

```
Select TRANSFER OPTION: TRANSFER FILE ENTRIES

 INPUT TO WHAT FILE: SCHOLAR
 TRANSFER FROM FILE: SCHOLAR
 TRANSFER DATA INTO WHICH SCHOLAR: RU
      1    FMPATIENT,23 A.
      2    FMPATIENT,23 H.
 CHOOSE 1-2:   1
 TRANSFER FROM SCHOLAR: FMPATIENT,23 H.
     WANT TO DELETE THIS ENTRY AFTER IT'S TRANSFERRED?  NO// YES
  ...EXCUSE ME, LET ME THINK ABOUT THAT A MOMENT.....
```

Data values are then merged such that no pre-existing values are overwritten but new values are added. The DATE OF BIRTH is added since the pre-existing value was null. Different subentries in the TOPICS Multiple are added to the pre-existing list of topics. The Inquire to File Entries option shows the result:

```
NAME:  FMPATIENT,23 A.  DATE OF BIRTH:  1872   SSN:  000-99-9999
   SUBJECT AREA:  PHILOSOPHY
 TOPICS: 23'S PARADOX
 TOPICS: PRINCIPLE OF EXCLUDED MIDDLE
 TOPICS: SET THEORY
 TOPICS: THEORY OF TYPES
 TOPICS: LIAR PARADOX
 TOPICS: AXIOM OF INFINITY

The entry for FMPATIENT,23 H. has been deleted.
```

Transferring Entries Between Files

You can use the Transfer File Entries option to move *all* or a *group* of entries from one file to an entirely separate file. To do this:

1. Answer the "INPUT TO WHAT FILE:" prompt and the "TRANSFER FROM FILE:" prompt with different file names.
2. Specify whether transferred entries should be added all as new, or whether they should be merged with existing entries.
3. Specify whether transferred entries should be deleted in the original file.
4. Specify which entries to transfer, by entering sort criteria.

For transfer to occur, the NAME fields (#.01) of both files *must* have matching LABEL fields and DATA TYPE fields. In this way, Fileman can identify corresponding entries. Values of the fields can then be transferred. Only those fields where the LABEL and DATA TYPE fields match will be transferred. Before the transfer is done, you are told which fields will have their data transferred.

The dialogue presented when transferring entries to another file is presented below. In this instance, you are transferring the contents of the SCHOLAR file to the NEW SCHOLAR file. The NEW SCHOLAR file already exists, and it does have some entries whose NAME field matches those in the SCHOLAR file.

```
Select OPTION: TRANSFER ENTRIES

Select TRANSFER OPTION: TRANSFER FILE ENTRIES

INPUT TO WHAT FILE: NEW SCHOLAR
TRANSFER FROM FILE: NEW SCHOLAR// SCHOLAR
'NAME' FIELDS, 'SSN' FIELDS, 'DATE OF BIRTH' FIELDS, 'SUBJECT
AREA'
FIELDS,'TOPICS' FIELDS, WILL BE TRANSFERRED

WANT TO MERGE TRANSFERRED ENTRIES WITH ONES ALREADY THERE? NO//
YES
WANT EACH ENTRY TO BE DELETED AS IT'S TRANSFERRED? NO
TRANSFER ENTRIES BY: NAME// <Enter>
START WITH NAME: FIRST// <Enter>
DEVICE: HOME// <Enter>
```

You can specify whether you want entries with the same NAME field to be merged when the transfers are made, or whether each transferred entry and subentry should become a distinct new entry in the target file. (In this case, the answer was YES to merge.) You can also specify whether or not the entries should be deleted from the "from" file as they are transferred. (In this case, the answer was NO to delete.)

A simple report is created that lists the entries that were transferred. You have the ability to route that list to a printer using the "DEVICE:" prompt.

In addition, you have considerable control over which entries are transferred. Your answers to the "TRANSFER ENTRIES BY:" and "START

WITH:" prompts select entries in the same way that you specify sort criteria when describing a print output.

REF: For more information on sort criteria and print output, see the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

For example, if you only wanted to transfer scholars born after 1900 to the NEW SCHOLAR file, you could answer the "TRANSFER ENTRIES BY:" prompt like this:

```
    TRANSFER ENTRIES BY: NAME// DATE OF BIRTH>1900
    WITHIN DATE OF BIRTH>1900, TRANSFER ENTRIES BY: <Enter>
```

NOTE: The Transfer File Entries option can be used to purge files. You can define a SCHOLAR ARCHIVE file containing a subset of the fields that are in the original file (perhaps the fields: NAME, SSN, and DATE OF BIRTH); you can then simply transfer into this separate file all or a selected group of entries from the original file, deleting the entries as they are transferred.

Transferring Entries into a New File

You can use the Transfer File Entries option to create a new file. To do this:

1. At the "INPUT TO WHAT FILE:" prompt, enter the name of a nonexistent file.
2. If you have programmer access, you will be prompted for the global location for the new file.
3. When you specify the file to transfer *from*, you can request that the data dictionary of that file be copied as the data dictionary for your new file.

Here is an example of using the Transfer File Entries option to create a new file:

```
INPUT TO WHAT FILE: SCHOLAR COPY
     Are you adding 'SCHOLAR COPY' as a new FILE?  No// Y
```

```
   FILE NUMBER: 16031// <Enter>

INTERNAL GLOBAL REFERENCE: ^DIZ(16031,// <Enter>
...SORRY, I'M WORKING AS FAST AS I CAN...
     A FreeText NAME Field (#.01) has been created.
TRANSFER FROM FILE: SCHOLAR
DO YOU WANT TO TRANSFER THE 'SCHOLAR' DATA DICTIONARY INTO YOUR
NEW FILE? YES
```

NOTE: You are asked the global reference question only if you have programmer access.

Answering YES copies (or "clones") the data definitions of the old file. Then, if the old file has had any templates created, you will be asked:

```
DO YOU WANT TO COPY 'SCHOLAR' TEMPLATES INTO YOUR NEW FILE?
```

Once you have created a new file with identical field and template descriptions, you can transfer entries into it. This is a method of copying a file.

Compare/Merge File Entries Option

The Compare/Merge File Entries option allows you to compare the data value of two entries before merging them into one entry. Furthermore, this option provides you with an opportunity to identify the data values from either entry that will be used to create the final merged entry. Both of the entries involved *must* be in the same file to use this option.

Comparing Entries

You can use the Compare/Merge File Entries option as a simple tool to compare entries. To do this:

1. Identify a file.
2. Identify the two entries to be compared.

3. Answer NO to the "MERGE ENTRIES AFTER COMPARING THEM?" prompt.

In this example below, two similar entries in the SCHOLAR file are used.

```
Select TRANSFER OPTION: COMPARE/MERGE FILE ENTRIES
COMPARE ENTRIES IN WHAT FILE: SCHOLAR
COMPARE SCHOLAR: RU
```

Fileman responds to this abbreviated response with the matching entries in the list that follows:

```
    1 FMPATIENT,23 A.
    2 FMPATIENT,23 H.
CHOOSE 1-2: 1
   WITH SCHOLAR: FMPATIENT,23 H.
```

NOTE: Use this option ONLY DURING NON-PEAK HOURS if merging entries in a file that is pointed-to either by many files, or by large files.

```
MERGE ENTRIES AFTER COMPARING THEM? No// <Enter>
```

Here you choose whether to simply compare entries or to actually merge them. If you are merging, the process of repointing entries in other files from the merged-from entry to the merged-to entry can be time-consuming or may create many tasked jobs. This is more likely if the file is pointed to by many files or if the files that point to it have many entries. In these situations, consider merging at times when your system is not busy.

```
DO YOU WANT TO DISPLAY ONLY THE DISCREPANT FIELDS?  NO// <Enter>
DEVICE: <Enter>                    RIGHT MARGIN:     80// <Enter>

COMPARISON OF SCHOLAR FILE ENTRIES   FEB 14, 2011  11:59  PAGE 1
SCHOLAR              FMPATIENT,23 A.              FMPATIENT,23 H.
-----------------------------------------------------------------

*** NAME             FMPATIENT,23 A.              FMPATIENT,23 H.

*** SSN              000-99-9999                  000-88-8888

*** SUBJECT AREA     PHILOSOPHY                   MATHEMATICS
*** DATE OF BIRTH   1872

Press RETURN to continue or '^' to exit:
```

The asterisks ("***") appearing in front of the field label indicate the entries contain different data in those fields. This simple report compares the data in each field in the two entries.

Merging Entries

You can use the Compare/Merge File Entries option to merge entries. To do this:

1. Identify a file.
2. Identify the two entries to be compared.
3. At the "MERGE THESE ENTRIES AFTER COMPARING THEM?" prompt, answer **YES**.
4. Choose which entry will supply the default values.
5. Choose to retain or delete the "merge from" record.
6. Optionally adjust, field-by-field, which entry supplies the default value.
7. Choose to proceed with the merge, summarize before merging, or re-edit the merge criteria.

The Merge process is described in detail via an example below:

```
MERGE THESE ENTRIES AFTER COMPARING THEM? NO// YES
    1 FMPATIENT,23 A.
    2 FMPATIENT,23 H.
```

After choosing a file and two entries to compare, enter YES at the "MERGE THESE ENTRIES AFTER COMPARING THEM? NO//" prompt to merge the entries as well as compare them.

You now specify which of the two entries will be used to supply the default values. The entry you choose will be the "merge to" entry that will contain the merged data.

```
NOTE: Records will be merged into the entry selected for the
default.
WHICH ENTRY SHOULD BE USED FOR DEFAULT VALUES (1 OR 2)? 1
 *** Records will be merged into FMPATIENT,23 A.
```

You next indicate the disposition of the "merge from" entry. You can choose to delete it or retain it unmodified. You can also redirect pointers that point to the "merged from" entry to point to the "merge to" entry. Free text pointers will *not* be redirected.

```
DO YOU WANT TO DELETE THE MERGED FROM ENTRY AFTER MERGING ? YES
DO YOU WANT TO REPOINT ENTRIES POINTING TO THIS ENTRY? YES
ENTER FILE TO EXCLUDE FROM REPOINT/MERGE:<Enter>
```

NOTE: You can also choose to exclude pointers from specified files in the repointing process. In this example (Error: Reference source not found), all pointers will be repointed.

```
DO YOU WANT TO DISPLAY ONLY THE DISCREPANT FIELDS?  NO// <Enter>
DEVICE: <Enter>
```

Press the **Enter** key at the "DEVICE:" prompt so that you can continue to control how the merge is to proceed. Since the merge is interactive, it is not appropriate to release control to a printer or other output device. While working at a keyboard printer is possible, it is not recommended, because you will not be able to monitor the selection of data values that is described below.

Fileman will display the data from the entries being merged. Brackets indicate the values that will be used to create the final merged entry. If the data values in both entries are the same, no brackets are shown. To start, the values for FMPATIENT,23 A. are bracketed because that entry was chosen as the default entry. To switch the value that will go into the merged entry, enter that field's number. The following example shows the initial defaults in brackets and the response to switching the value for the SSN field:

```
COMPARISON OF SCHOLAR FILE ENTRIES  FEB 14, 2011  11:59  PAGE 1
```

```
SCHOLAR                 [FMPATIENT,23 A.]        FMPATIENT,23 H.
--------------------------------------------------------------------

1.  NAME                    [FMPATIENT,23 A.]
FMPATIENT,23 H.

2.  SSN                     [000-99-9999]               000-88-8888

3.  SUBJECT AREA            [PHILOSOPHY]                MATHEMATICS
AND PHILOSOPHY

4.  DATE OF BIRTH           [1872]

Default is enclosed in brackets, e.g., [FMPATIENT,23 A.]
Enter 1-4 to change a default value, ^ to exit report, RETURN to
continue: 2


COMPARISON OF SCHOLAR FILE ENTRIES  FEB 14, 2011  11:59   PAGE 1
SCHOLAR                     [FMPATIENT,23 A.]
FMPATIENT,23 H.
--------------------------------------------------------------------

1.  NAME                    [FMPATIENT,23 A.]
FMPATIENT,23 H.

2.  SSN                     000-99-9999                [000-88-
8888]

3.  SUBJECT AREA            [PHILOSOPHY]                MATHEMATICS
AND PHILOSOPHY

4.  DATE OF BIRTH           [1872]

Default is enclosed in brackets, e.g., [FMPATIENT,23 A.]
Enter 1-4 to change a default value, ^ to exit report, RETURN to
continue: <Enter>
```

NOTE: The DATE OF BIRTH field from FMPATIENT,23 H. is bracketed because the default entry has no value for that field; it is null. When the default ("merged to") entry has no data in a field, data will always be brought over from the "merged from" entry. You have no control over this aspect of the merging. Even if you attempt to switch the data value to the null value and remove the brackets, the data will still be brought into the "merged to" field.

If there is more than one screen of field information, Fileman will display additional screens. In this case, there is one field that is a Multiple. The Compare/Merge File Entries option neither displays data from WORD-PROCESSING or Multiple fields nor allows the selection of data for the final "merge to" entry. The number of subentries in the Multiples of the two fields is displayed:

```
COMPARISON OF PATIENT FILE ENTRIES   FEB 14, 2011  11:59  PAGE 2
SCHOLAR            [FMPATIENT,23 A.]          FMPATIENT,23 H.
-------------------------------------------------------------------
  NOTE: Multiples will be merged into the target record.

1. "TOPICS"       " 5 entries"             " 4 entries"

Enter RETURN to continue: <Enter>
```

The merging of data for a WORD-PROCESSING field or for a Multiple and its subfields is done in the same way that the Transfer File Entries option does it.

Although Fileman is ready to perform the merge at this time, you are prompted with other options—just in case you are not ready. In order to cancel the merge, enter the caret ("^") at the "ACTION:" prompt to exit.

```
OK.  I'M READY TO DO THE MERGE.

       Select one of the following:
             P PROCEED to merge the data
             S SUMMARIZE the modifications before proceeding
             E EDIT the data again before proceeding
 ACTION:
```

These three options are:

P—PROCEED to merge the data
S—SUMMARIZE the modifications before proceeding
E—EDIT the data again before proceeding

PROCEED

If you enter P or PROCEED at the "ACTION:" prompt, then Fileman will proceed to merge the data specified:

```
ACTION: PROCEED to merge the data.

 I will now merge all subfiles in this file ...
 This may take some time, please be patient.

 I will now repoint all files that point to this entry ...
 This may take some time, please be patient.

 Gathering files and checking 'PT' nodes

 Merging entries

 Merge complete

 Deleting From entry
```

The merging of the data is now complete. Be careful when using this option because the "merge from" entry can be deleted and data could be lost.

SUMMARIZE

To review the changes that will be made to the "merge to" entry, enter an S or SUMMARIZE at the "ACTION:" prompt before proceeding to merge. The following will be shown:

```
ACTION:  SUMMARIZE the modifications before proceeding

 SUMMARY OF MODIFICATIONS TO FMPATIENT,23 A.

 FIELD   OLD VALUE  NEW VALUE

 -------------------------------------------------

 2.  SSN   000-99-9999  000-88-8888

 4.  DATE OF BIRTH    1872

   NOTE: Multiples will be merged into the target record

 Enter RETURN to continue: <Enter>
```

EDIT

Enter an E or EDIT at the "ACTION:" prompt to change the decisions made earlier. If you choose this action, you will again be shown the values in both entries, with your current selections in brackets, and will have the opportunity to switch the data going into the "merge to" entry.

Namespace Compare Option

The Namespace Compare option allows you to compare data dictionary and file contents across VISTA instances. You select the UCI that you want to compare to your UCI. Then you select the file or range of files you want to compare. Finally, you specify whether you want the data dictionary, the file contents, or both to be compared.

The following example shows the comparison of both Data Dictionary and File Entries for the Language file. The current UCI is identified. You select the file to be compared, the target UCI, and the scope of the comparison.

The report has two columns, one for each UCI. It first lists differences between the Data Dictionaries. In this case, the target UCI does not have a “B” Index. Then File Entries are compared. When a difference is identified, the value of the entry’s .01 field and its IEN is shown as a heading followed by fields containing different values along with their values. Here, the DATE INPUT field has a value of “Q” in the home UCI and no value in the target UCI. Finally, this example lists entries that exist in the home UCI but are missing in the target UCI.

Sample Namespace Compare Session

```
Select OPTION: TRANSFER ENTRIES

Select TRANSFER OPTION: ?
    Answer with TRANSFER OPTION NUMBER, or NAME
   Choose from:
   1            TRANSFER FILE ENTRIES
   2            COMPARE/MERGE FILE ENTRIES
   3             NAMESPACE COMPARE

Select TRANSFER OPTION: 3  NAMESPACE COMPARE
  UCI: /home/dev1/g/db.gld

 START WITH What File: LANGUAGE         (533 entries)
      GO TO What File: LANGUAGE// <Enter>        (533 entries)
Compare to what UCI: /home/sample/emptyENV/g/mumps.gld

     Select one of the following:

          1         DATA DICTIONARY ONLY
          2         FILE ENTRIES ONLY
          3         DATA DICTIONARY AND FILE ENTRIES

Enter response: 3// <Enter>   DATA DICTIONARY AND FILE ENTRIES
DISPLAY COMPARISON ON
DEVICE: HOME//<Enter>  TELNET

MAR 13, 2013  FILEMAN.MUMPS.ORG
  UCI: /home/dev1/g/db.gld              UCI:
/home/sample/emptyENV/g/mumps.gld
----------------------------------------------------------------


                        DATA DICTIONARY #.85 (LANGUAGE)
 INDEX: B^Regular new-style B Index

                        ENTRIES IN FILE #.85 (LANGUAGE)
                             LANGUAGE: FRENCH (#4)
  DATE INPUT: Q
 LANGUAGE #8: ABKHAZ
 LANGUAGE #9: ACHINESE
 LANGUAGE #13: ACOLI
 LANGUAGE #14: ADANGME
 LANGUAGE #15: ADYGHE
 LANGUAGE #16: AFAR
 LANGUAGE #17: AFRIHILI
 LANGUAGE #19: AFRIKAANS

Type <Enter> to continue or '^' to exit: ^
```

You cannot compare a GT.M database against a Cache database.

Under GT.M, your Global Directory file must contain only absolute path references to use the Namespace Compare option. If you Global

Directory uses relative paths or environmental variables, this option will not work properly.

Chapter 12: Extract Tool

Using the Extract Tool, you can move or copy data from logical records in Fileman files to a new Fileman file. This new file may either *permit* users to modify its contents or *prevent* users from modifying its contents and can be available for online inquiries and print processes. If this new file is used to store archived data, any options and utilities that create new entries or update existing entries are restricted. Options and utilities that update the data dictionary are also restricted.

Extract Overview

The following is an overview of the process of using the Extract Tool to extract entries from a file:

1. Identify the files and fields from which to extract data by using information in the data dictionary listings.
2. Build a destination file by creating a new field for each field in the source file.
3. Select the source file entries from which data will be extracted by creating a SEARCH/SORT template.
4. Select the fields from which data will be extracted by creating an EXTRACT template.
5. Move the extracted data to the destination file by using the Update Destination File option.
6. Purge the selected entries from the active database.

Important Items to Note

Before beginning the extract process, consider each of the following important facts about the Extract Tool:

- An extract activity is file-specific, not user-specific. Anyone with access to the file and the Extract Tool options can complete or change an existing extract activity.

- When the extracted data is moved to the destination file, the source entries in the primary file are blocked from selection.
- A Subfile cannot be extracted by itself. At least one field from every Multiple level *above* the Subfile *must* also be extracted. If no field is extracted at the next higher level, the .01 field at that level will automatically be extracted.
- You may want to extract Identifiers and/or KEY fields from the source file to the destination file, so that records in the destination file can be uniquely identified.
- An EXTRACT template is the only type of PRINT template that can be used by the Update Destination File option.
- The ARCHIVAL ACTIVITY file (#1.11) contains a brief history that describes who performed the various extract activity steps and when the steps were completed.
- The extract activity can be canceled at any time before the Purge Extracted Entries option is used.
- The Purge Extracted Entries option deletes all of the source data in the primary file from which you extracted data.
- Selected entries cannot be purged until they have been moved to the destination file.
- A second extract from a file cannot be performed until the active extract activity has been completed, either by purging or canceling.

Source File

The term source file represents the primary file and any other files that can be referenced by extended pointers. The primary file is the starting file from which you will extract your data. The term extract field refers to any field in the source file.

Destination File

The term "destination file" represents the Fileman file that stores the extracted data. The destination file can be located anywhere on the network that is recognized by the system. To create this file, you select either of two Fileman options: Modify File Attributes or the Extract Tool's Modify Destination File.

For each extract field in the source file, a corresponding field in the destination file *must* exist. Certain DATA TYPE field values can optionally be resolved to external form before moving the data to the destination file. For example, data extracted from a DATA TYPE field of POINTER TO A FILE can be moved to a FREE TEXT-type field in the destination file, if external form of data is moved; or such data can be moved to a NUMERIC-type field, if the internal value is moved.

REF: For more information, see the "Mapping Information" section.

The destination file uses a file level attribute called ARCHIVE FILE. The following is a description of this flag:

- YES—This is an archive file and users *cannot* modify or delete the data or the data dictionary. Any data dictionary changes may invalidate the archived data.
- NO (or null)—There are no restrictions on the file.

If you need to update an archive file's data dictionary, you *must* convert the old data to the new data dictionary format.

Updates to an archive file will be allowed only through the extract option, Update Destination File option, or through the programmer entry point EXTRACT^DIAXU.

REF: For more information on the Extract Tool API (EXTRACT^DIAXU), see the Fileman *Programmer Manual*.

Only Regular, KWIC, and Soundex-type cross-references are recommended for archive files. No other types of cross-references should be created.

If you are building a destination file that will store archived data, set the

ARCHIVE FILE flag to YES (do this with the Modify Destination File option). Setting the ARCHIVE FILE flag to YES prevents users from modifying or deleting the data in the file or the file's data dictionary while using Fileman options or programmer calls. Users are also prevented from deleting file entries while using Fileman options or programmer calls.

Mapping Information

Mapping information identifies the relationship between the data in the source file and the data in the destination file. When you create your EXTRACT template, you will enter the name of the field in the source file and identify its intended location in the destination file. You will need to ensure that the DATA TYPE field value of the field in the destination file is compatible with the DATA TYPE field value of the extract field. The compatibility of the DATA TYPE field values is validated when the fields are specified during template creation.

The following table recommends the DATA TYPE field values to use, depending on the DATA TYPE field value of the extract field:

DATA TYPE Field Value of Extract Field	DATA TYPE Field Value of Destination Field
DATE/TIME	1) DATE/TIME, internal form of data is moved. 2) FREE TEXT, external form of data is moved.
NUMERIC	NUMERIC or FREE TEXT.
SET OF CODES	1) FREE TEXT, if external form of the SET OF CODES is moved. 2) SET OF CODES, if internal form of the SET OF CODES is moved. User *must* make sure the SET OF CODES fields are identical in both the source file and the destination file data dictionaries.
FREE TEXT	FREE TEXT.

DATA TYPE Field Value of Extract Field	DATA TYPE Field Value of Destination Field
WORD-PROCESSING	WORD-PROCESSING.
COMPUTED	FREE TEXT, DATE/TIME, or NUMERIC.
POINTER TO A FILE	1) NUMERIC, if internal form of data is moved. 2) Non-pointer field type (FREE TEXT, NUMERIC, or DATE/TIME), if external form of data is moved.
VARIABLE-POINTER	1) Non-pointer field type, if external form of data is moved; (if the .01 fields of the pointed-to files have different DATA TYPE field values, DATA TYPE field values of destination field should be FREE TEXT). 2) FREE TEXT, if internal form of data is moved.
MUMPS	MUMPS.
Multiples	Multiples.
Backward Pointers	Multiples.

Here are additional guidelines that you *must* follow while creating your destination file:

- If you are extracting a **SET OF CODES**-type field and you are mapping it to a **FREE TEXT**-type field, use a maximum length of the same—or greater than—length as the longest external value in the SET OF CODES field. If you are mapping the SET OF CODES-type field to a SET OF CODES-type field, create the corresponding field in the destination file, using the same specifications as the extract field.
- If you are extracting DATA TYPE field values of any of the following:
    - FREE TEXT
    - DATE/TIME

    - NUMERIC
    - WORD-PROCESSING
    - MUMPS

  Create the corresponding field in the destination file, using the same specifications as the extract field.
- If you are extracting a **Multiple**-type field, create the corresponding field in the destination file as a Multiple-type field. Multiples in the source file are moved to Multiples in the destination file, following the DATA TYPE field value recommendations. The structure of the Multiple in the destination file should be the same as that in the source file down to the lowest level Multiple that you extract. When extracting data in a Subfile, at least one field from every Multiple level above the Subfile *must* also be extracted. If you do not specify a field to extract at a higher level, the .01 field at that level will automatically be extracted.
- If you are extracting a **Backward Extended Pointer**-type field, create the corresponding field in the destination file as a Multiple. The Extract Tool resolves Backward Pointers. Thus, their values are moved to Multiples in the destination file.
- If the field you are extracting has an **OUTPUT transform**, make sure the **INPUT transform** of the destination field can receive the data in the format generated by the OUTPUT transform.

ARCHIVAL ACTIVITY File

To learn about the status of an extract activity you can enter a question mark at most of the prompts in the Extract Tool options. Using the Inquire to File Entries option on the ARCHIVAL ACTIVITY file (#1.11) will yield information about past or pending activities. Those activities created by the Extract Tool are referred to as extract activities. The amount of information displayed depends on the status of the extract activity. The ARCHIVAL ACTIVITY file contains the following information:

- The DUZ of the individual performing the extract activity.
- The status of the extract activity (e.g., EDITED or UPDATED).
- The dates on which the activities were performed.
- The number of entries extracted.

- The source file number.
- The SEARCH/SORT and PRINT templates used in the extract activity.

For Fileman 20.0 and later, the ARCHIVAL ACTIVITY file contains data about both archiving and extract activities. A file can have only *one* active activity at a time—either an archiving activity or an extract activity. You can only select an extract activity from the Extract Tool options. When you use the Inquire to File Entries option, the word EXTRACT will appear for all extract activities.

Extract Steps

The order of the options on the Extract Tool submenu reflects the sequence of steps in which you ordinarily perform your extract activity. To access to the Extract Tool options, start at the Other Options submenu. Here's a sample of the dialogue that you will encounter.

```
Select OPTION: OTHER OPTIONS
Select OTHER OPTION: EXTRACT DATA TO FILEMAN FILE
Select EXTRACT OPTION: ?
 ANSWER WITH EXTRACT OPTION NUMBER, OR NAME CHOOSE FROM:
     1              SELECT ENTRIES TO EXTRACT
     2              ADD/DELETE SELECTED ENTRIES
     3              PRINT SELECTED ENTRIES
     4              MODIFY DESTINATION FILE
     5              CREATE EXTRACT TEMPLATE
     6              UPDATE DESTINATION FILE
     7              PURGE EXTRACTED ENTRIES
     8              CANCEL EXTRACT SELECTION
     9              VALIDATE EXTRACT TEMPLATE
```

Select Entries to Extract Option (1 of 9)

The Select Entries to Extract option initiates the extract activity. In this option the entries are selected and stored in a template and an entry in the ARCHIVAL ACTIVITY file is created. Entries are selected in the same manner as the Search File Entries option.

REF: For guidance on selecting entries, see the "Search" chapter of the *Fileman User Manual*.

The Select Entries to Extract option performs the following functions:

1. During the search phase, the search criteria for selecting entries is specified and *must* be stored in a template.

   NOTE: If you want to extract a subentry contained in a Multiple field, you *must* extract the entire entry.
2. During the sort phase—which is indicated by the "SORT BY" prompt—you can enter additional restrictions on the entries to be selected. If no further restrictions are required, simply accept the defaults provided at the "SORT BY" and "START WITH" prompts.
3. During the print phase—which is indicated by the "PRINT FIELD" prompt—Fileman gathers the entries specified in the search and sort phases and adds the internal entry numbers of the selected entries to the SEARCH template. Although specifying print fields is not required, the print process *must* be run to completion. Simply press the **Enter** key at the "PRINT FIELD" prompt or specify actual fields to print a report of identifying information for the extracted records.

In the sample dialogue that follows, notice the sequence in which the search, sort, and print prompts appear:

```
EXTRACT FROM WHAT FILE: CHANGE

  -A- SEARCH FOR CHANGE FIELD: .01 <Enter>  NO.
  -A- CONDITION: LESS THAN
  -A- LESS THAN: 900

  -B- SEARCH FOR CHANGE FIELD: <Enter>

  IF: A// <Enter>  NO. LESS THAN 900

STORE RESULTS OF SEARCH IN TEMPLATE: ZZTEST TEMPLATE
 Are you adding 'ZZTEST TEMPLATE' as a new SORT TEMPLATE?  No//
Y

SORT BY: VERSION
START WITH VERSION: FIRST// <Enter>
```

```
  WITHIN VERSION, SORT BY: **<Enter>**
FIRST PRINT FIELD: .01 <Enter> NO.
THEN PRINT FIELD: VERSION
THEN PRINT FIELD: PROGRAMMER
THEN PRINT FIELD: <Enter>
HEADING: CHANGE EXTRACT SEARCH Replace <Enter>
DEVICE: <Enter>
```

The resulting output looks like:

```
CHANGE EXTRACT SEARCH                AUG 30, 1992  10:59   PAGE 1
NO.          VERSION           PROGRAMMER
------------------------------------------------------------------

101          17.10             FMPROGRAMMER,25
102          17.32             FMPROGRAMMER,26
103          17.35             FMPROGRAMMER,26
    3 MATCHES FOUND.
```

After you use this option, Fileman marks the ARCHIVAL ACTIVITY file entry with a status of SELECTED. If an unfinished extract activity exists for a file and you select this same file for a subsequent extract activity, you will see the following message:

```
    There is already an outstanding extract activity.
    Please finish it or CANCEL it.
```

Since the ARCHIVAL ACTIVITY file maintains a record of both your extract and archiving activities, you will see the italicized word *archiving* whenever the outstanding file activity is an archiving one. To add or delete entries from the SEARCH/SORT template you just created, use the Add/Delete Selected Entries option.

Add/Delete Selected Entries Option (2 of 9)

When you wish to add entries to the extract activity or you wish to delete an entry or entries, use the Add/Delete Selected Entries option. This option provides an easy way to eliminate undesired entries or to add needed ones to your list of entries to extract. Like the Inquire to File Entries option, this option displays a selected entry and then asks if you wish to delete or add the entry. If you modify the list, then the activity's status in the ARCHIVAL ACTIVITY file changes from SELECTED to EDITED.

You can only use this option to modify your list before the entries are moved to the destination file. If you need to change the extract activity list after the destination file is updated, you will need to cancel the extract activity and start a new extract activity.

To use the Add/Delete Selected Entries option, select the extract activity you wish to modify by entering the archival activity number, source file number, or source file name. Then select the entry to be added or deleted. The following dialogue depicts the sequence you will follow when adding an entry to the extract activity:

```
Select EXTRACT OPTION: ADD/DELETE SELECTED ENTRIES
Select EXTRACT ACTIVITY: ?
  ANSWER WITH ARCHIVAL ACTIVITY ARCHIVE NUMBER, OR FILE
CHOOSE FROM:
  3    CHANGE  08-30-92  SELECTED     SELECTOR:FMEMPLOYEE,J
       EXTRACT

Select EXTRACT ACTIVITY: 3 <Enter>  08-30-92  SELECTED
  SELECTOR:FMEMPLOYEE,J  EXTRACT

Select CHANGE NO.: 330
NO.:  330                        VERSION:  17.09
 PROGRAMMER:  FMPROGRAMMER,27          ROUTINE:  DIL2
 DATE CHANGED:  OCT 24, 1995

ADD this entry TO the EXTRACT SELECTION?  YES// <Enter>
```

NOTE:

- Entering two question marks ("**??**") at the "Select EXTRACT ACTIVITY:" prompt displays a list of file entries.
- The phrase "*on EXTRACT list*" appears next to those entries that are currently part of the extract activity.
- The question "DELETE this entry...?" appears whenever you select an entry that is currently on the extract list.
- "ADD this entry...?" appears whenever you select an entry that is not among the items on the list.

Print Selected Entries Option (3 of 9)

To display the list of entries you have selected, use the Print Selected Entries option. This option uses the standard Fileman interface for printing.

REF: For guidance on printing entries, see the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

The example that follows depicts the type of dialogue you will encounter when printing a list of entries to be extracted:

```
Select EXTRACT OPTION: PRINT SELECTED ENTRIES
Select EXTRACT ACTIVITY: 3 <Enter>  CHANGE 08-30-92  EDITED
 SELECTOR:FMEMPLOYEE,J  EXTRACT

Enter a regular Print Template name or fields you wish to see
printed on this report of records to be extracted.

FIRST PRINT FIELD: [ZZTEST TEMPLATE
```

The output looks like:

```
CHANGE EXTRACT ACTIVITY              AUG 30, 1992  11:09   PAGE 1
NO.                   VERSION            PROGRAMMER
----------------------------------------------------------------
101                   17.10              FMPROGRAMMER,25
102                   17.32              FMPROGRAMMER,25
103                   17.35              FMPROGRAMMER,25
330                   17.09              FMPROGRAMMER,30
```

Modify Destination File Option (4 of 9)

You can use either the Modify File Attributes option or the Modify Destination File option when you are ready to create the destination file that will receive your extracted data. You can also use these options to correct discrepancies that you noticed while you were building your EXTRACT template. The two options are nearly identical. However, one major difference exists: the Modify Destination File option prompts for a new file attribute: ARCHIVE FILE (see the "Destination File" topic previously described in this chapter). If you use the Modify File Attributes option to create the destination file, you will need to access the Modify Definition File option to set the ARCHIVE FILE flag.

Here is a sample of the type of dialogue that you will encounter when modifying your destination file:

```
Select EXTRACT OPTION: MODIFY DESTINATION FILE

This option allows you to build a file which will store data
extracted from other files.  When creating fields in the
destination file, all data types are selectable.  However, only
a
few data types are acceptable for receiving extracted data.

Please see your User Manual for more guidance on building the
destination file.

MODIFY WHAT FILE: CHANGE EXTRACT
```

From this point on, you will see the usual dialogue while creating a new file and creating fields.

Once you have finished creating your destination file, you will see the dialogue that follows:

```
Select FIELD: <Enter>
ARCHIVE FILE? NO// ?

    Enter either 'Y' or 'N'
```

```
ARCHIVE FILE? NO// ??
    'YES' will not allow modifications or deletions of data or
data
dictionary
    'NO'  will place no restrictions on the file.

ARCHIVE FILE? NO// <Enter>

Select EXTRACT OPTION:
```

Create Extract Template Option (5 of 9)

When selecting destination fields for data to be extracted into, keep in mind that the INPUT transforms of the destination fields are executed for each field value. For an extracted record, the value of each field in the record is tested against the INPUT transform of its destination field. If any field fails the INPUT transform, the extract for the entire record will fail. Make sure the INPUT transforms on the destination fields are appropriate for the data you will be extracting.

NOTE: If you are extracting a **Sub**record using the EXTRACT^DIAXU entry point and its FILING_LEVEL parameter, and a value fails the INPUT transform, only the extract of the Subrecord will fail.

When you are ready to build an EXTRACT template, you *must* select the Create Extract Template option. Using this option, you will identify not only the field you wish to extract from the source file but also its corresponding field in the destination file. The EXTRACT template is the only type of PRINT template used in the Update Destination File option.

Building an EXTRACT template requires entering valid field numbers or field names at the "EXTRACT FIELD" prompt. Since Fileman stores EXTRACT templates in the PRINT TEMPLATE file (#.4), this option uses the term "PRINT TEMPLATE" instead of "EXTRACT TEMPLATE" in the dialogue. For each extract field that you identify in the source file, at the "MAP TO" field prompt, enter the destination file field name or field number that will receive the data. Only those fields defined in the EXTRACT template will appear in the destination file.

Keep in mind that the value of each field in an extracted record is tested against the INPUT transform of its destination field. If any value fails its destination field's INPUT transform, the extract for the entire record will fail. Make sure the INPUT transforms on the destination fields are appropriate for the data you will be extracting.

NOTE: If you are extracting a **Sub**record using the EXTRACT^DIAXU entry point and its FILING_LEVEL parameter, and a value fails the INPUT transform, only the extract of the Subrecord will fail.

When you arrive at the "STORE EXTRACT LOGIC IN TEMPLATE:" prompt, enter the name that you wish to assign to your new EXTRACT template. To edit an existing EXTRACT template, on the other hand, simply enter its name at the "FIRST EXTRACT FIELD:" prompt—using the following format:

```
"[Extract templatename"
```

Here is a sample of the dialogue that you will encounter when you are ready to build an EXTRACT template:

```
Select EXTRACT OPTION: **CREATE** EXTRACT TEMPLATE

This option lets you build a template where you specify fields
to extract and
their corresponding mapping in the destination file.

For more detailed description of requirements on the destination
file, please
see your Fileman User Manual.

OUTPUT FROM WHAT FILE: **CHANGE**
DESTINATION FILE: **CHANGE EXTRACT**

FIRST EXTRACT CHANGE FIELD: **.01 <Enter>**  NO.
MAP NO. TO CHANGE EXTRACT FIELD: **.01 <Enter>**  NO.

 THEN EXTRACT CHANGE FIELD: **VERSION**
 MAP VERSION TO CHANGE EXTRACT FIELD: **VERSION**

 THEN EXTRACT CHANGE FIELD: **PROGRAMMER**
 MAP PROGRAMMER TO CHANGE EXTRACT FIELD: PROGRAMMER

STORE EXTRACT LOGIC IN TEMPLATE: CHANGE EXTRACT
  Are you adding 'CHANGE EXTRACT' as a new PRINT TEMPLATE?  No//
YES
```

While you are creating your EXTRACT template, Fileman performs a few validation checks. Inspecting the extract field and its corresponding field in the destination file, Fileman checks to see if both fields are compatible in several important areas, including data type, minimum length, maximum length, minimum values, maximum values.

If a discrepancy exists, Fileman will display an error message such as the following statement:

```
    PROGRAMMER field in CHANGE EXTRACT file should have a
    maximum length of at least 30 characters.
```

After Fileman displays an error message about your destination field, you can continue building your template. You will not, however, be able to update the destination file until you have corrected the problem.

Here is the warning that you will see when any source field and its corresponding destination field fail one of the validation checks:

```
    THE DESTINATION FILE DATA DICTIONARY SHOULD BE MODIFIED
    PRIOR TO ANY MOVEMENT OF EXTRACT DATA!
```

At any "MAP 'FIELD NAME' TO 'FILE NAME' FIELD:" prompt, entering two question marks (" **??**") yields a list of the selectable fields in the destination file. The list gets shorter as fields are selected to ensure that no two extract fields map information to a single field in the destination file.

Update Destination File Option (6 of 9)

Once you have used the Update Destination File option, the extracted data from the source file is moved to the destination file. After you enter the name of the EXTRACT template that you wish to use, Fileman makes sure the template's mapping information is correct and acceptable and then populates the destination file, adding entries as new records. Fileman will not, however, check to see if any of those records to be moved already exist in the destination file. Since this two-step process can be quite time-consuming, it can be queued at the "DEVICE:" prompt.

Here is a sample of the dialogue:

```
Select EXTRACT OPTION: UPDATE DESTINATION FILE
Select EXTRACT ACTIVITY: 3 <Enter>  CHANGE  08-31-92     EDITED
SELECTOR:FMEMPLOYEE,J   EXTRACT

You MUST enter an EXTRACT template name.  This EXTRACT template
will be used to populate your destination file.
PRINT TEMPLATE: CHANGE EXTRACT <Enter>      **EXTRACT**    (AUG
30,1992)       USER #2  FILE #16000

Excuse me, this will take a few moments...
Checking the destination file...

If entries cannot be moved to the destination file, an exception
report will be printed.

Select a device where to print the exception report.

QUEUEING to this device will queue the Update process.
EXCEPTION REPORT DEVICE: QUEUE TO PRINT ON
DEVICE: PRINTER
```

After the destination file has been updated, Fileman changes the extract activity status from SELECTED or EDITED to UPDATED DESTINATION FILE. At this point, the entries from the source file are no longer available on lookups. This protective measure prevents you from attempting to edit the selected source file entries so that they contain the same data as the corresponding destination file entries.

The following Exception Report is printed when the Extract Tool fails to move all of the data in a source entry into the destination file. A failed INPUT transform is one possible cause of such a failure. In this case, the incomplete entry in the destination file is deleted. The source entry is not locked and its internal entry number is deleted from the extract list. The total number of entries extracted is reduced by the total numbers of entries appearing on the exception report.

```
EXTRACT  ACTIVITY EXCEPTION REPORT       JUN 27,1996   PAGE: 1
----------------------------------------------------------------

EXTRACT ACTIVITY: 9          ARCHIVER: FMEMPLOYEE,J

THE FOLLOWING ENTRIES IN THE 'TEST' FILE WERE NOT MOVED BY THE
EXTRACT TOOL

Entry # 9 was NOT processed because:
  The value 'NEW' for field FTEXT MULT LABEL in FTEXT MULT SUB-
FIELD in file TEST is not
valid.

Enter # 30 was NOT processed because:
  The value 'NEW' for field FTEXT MULT LABEL in FTEXT MULT SUB-
FIELD in file TEST is not
valid.

*** PLEASE KEEP THIS FOR FUTURE REFERENCE ***
```

The following is a list of recommended steps to take when an exception report is printed:

1. Finish the active extract activity by purging or canceling.
2. Determine the problem with the source entry and fix it.
3. If there are several entries on the exception report, start another extract activity. Your SEARCH/SORT template can be reused to use the same search specifications.
4. Adjust the extract list to match the list of entries on the exception report by using the Add/Delete Selected Entries option.
5. Proceed as before.

For exceptions caused by INPUT transforms, keep in mind that the value of each field in an extracted record is tested against the INPUT transform of its destination field. If any value fails its destination field's INPUT transform, the extract for the entire record will fail. Make sure the INPUT transforms on the destination fields are appropriate for the data you will be extracting.

NOTE: If you are extracting a **Sub**record using the EXTRACT^DIAXU entry point and its FILING_LEVEL parameter, and a value fails the INPUT transform, only the extract of the Subrecord will fail.

Purge Extracted Entries Option (7 of 9)

If you have DELETE access to the primary file, you can use the Purge Stored Entries option to delete extracted data from the primary file (our example is the CHANGE file). After you have purged your entries, Fileman will update the ARCHIVAL ACTIVITY file. If you attempt to purge an extract activity that lacks the status UPDATED DESTINATION FILE, you will encounter the following message:

```
Data has NOT YET been moved to the destination file!
```

When purging extracted data, you will encounter a dialogue much like the one that follows:

```
Select EXTRACT OPTION: PURGE EXTRACTED ENTRIES
Select EXTRACT ACTIVITY: 3 <Enter>  CHANGE  08-30-92     UPDATED
DESTINATION FILE   SELECTOR:FMEMPLOYEE,J     EXTRACT
```

If the source file has fields from other files pointing to it, the Extract Tool tells you:

```
The records about to be purged should not be 'pointed to' by
other
records to maintain database integrity.

This option will DELETE DATA from both CHANGE
and from the ARCHIVAL ACTIVITY file.
Are you sure you want to continue? NO// YES

The entries will be deleted in INTERNAL NUMBER order.

<< 4 ENTRIES PURGED >>
```

As you can see, entering a YES response to the "Are you sure you want to continue? NO//" prompt deletes the entries **immediately**!

Cancel Extract Selection Option (8 of 9)

You can cancel an extract activity any time before the entries are purged by using the Cancel Extract Selection option. If the extract activity status is UPDATED DESTINATION FILE—meaning the entries have already been moved to the destination file—you'll see a warning notice. At this point, you can roll back or delete the new entries that were created while using the Update Destination File option.

After you have canceled an extract activity, Fileman deletes the ARCHIVAL ACTIVITY file reference to the extract activity. In addition, you will once again be able to gain access to all of those source entries that Fileman locked during the update of your destination file. If you wish to extract data without purging the source entries, cancel the extract activity to unlock the selected entries in the source file.

You will encounter the following dialogue while canceling an extract activity:

```
Select EXTRACT OPTION: CANCEL EXTRACT SELECTION
Select EXTRACT ACTIVITY: CHANGE <Enter>  3 CHANGE 08-31-92
  UPDATED DESTINATION FILE      SELECTOR:FMEMPLOYEE,J EXTRACT

Are you sure you want to CANCEL this EXTRACT ACTIVITY? NO// ??
    Enter YES to stop this activity and start again from the
beginning.

Are you sure you want to CANCEL this EXTRACT ACTIVITY?  NO// YES

This extract activity has already updated the destination file.

Delete the destination file entries created by this extract
activity? NO// ??

     Enter YES to rollback the destination file to its state
before
the update.

Delete the destination file entries created by this extract
activity? NO// <Enter>

>>> DONE <<<
```

If you wish to cancel an extract selection so that you can start over, enter YES at the "Delete the destination file entries..." prompt. Entering YES prevents you from sending a duplicate set of entries to the destination file. If, on the other hand, you simply want to cancel the extract selection, pressing the **Enter** key at the prompt unlocks the source entries and retains the destination file entries.

Validate Extract Template Option (9 of 9)

After you have corrected any discrepancies that Fileman might have pointed out while you were creating an EXTRACT template, you can use the Validate Extract Template option to quickly check your EXTRACT template's mapping information. The Validate Extract Template option will *not* alter anything in your template. Here is a sample dialogue:

```
Select EXTRACT OPTION: VALIDATE EXTRACT TEMPLATE
Select EXTRACT TEMPLATE: CHANGE <Enter>   EXTRACT
**EXTRACT**
       (AUG 30, 1992)         USER #2  FILE #16000

Excuse me, this will take a few moments...
Checking the destination file...

Template looks OK!
```

Chapter 13: Filegrams

NOTE: In order to use the full capabilities of Fileman's Filegram procedures, Kernel 6.5 or later *must* be installed on your system.

Filegrams are a feature in Fileman intended for use by System Managers, Package Developers, and Programmers.

A filegram is a process that moves a record (also called an entry) from a file on one computer system to a duplicate file on another *independent* computer system. An independent computer system is defined as a system having its own database. Sending data from the "live" account at a medical center to a "test" account at the same medical center is an example of moving a filegram locally. Sending data from a computer in the San Francisco Medical Center to a computer in the Salt Lake City Medical Center is an example of moving a filegram remotely.

The records you can move by a filegram can be either physical (stored in one file) or logical (related fields stored in different files). Logical records are loaded into filegrams by using Fileman's relational navigational syntax (**:**).

For successful filegram installation, the recipient system *must* have the following:

- Kernel 6.5 or later.
- The FILEGRAM HISTORY file (#1.12).
- The Filegram key to the Filegram submenu.
- A file structure that matches the one reflected by the entry in the filegram. This is the structure existing on the sender's system.

FILEGRAM-type Templates

The Filegram process requires you to create an OUTPUT template, which is stored in the PRINT TEMPLATE file (#.4) along with regular PRINT templates. Fileman recognizes the FILEGRAM-type and regular PRINT templates as OUTPUT templates, but their similarity ends there.

NOTE: Since FILEGRAM-type templates are stored in the PRINT TEMPLATE file, the dialogues you encounter in the Filegram process will refer to a FILEGRAM-type template as a PRINT template.

Regular PRINT templates already created are screened so that you cannot accidentally replace an existing PRINT template with a FILEGRAM-type template.

Filegram and Archiving Relationship

FILEGRAM-type templates are the only kind of template allowed in the archiving process.

REF: For a full description of archiving in Fileman, see the "Error: Reference source not found" section.

Using Filegrams

Here is a summary of the basic steps needed to send and install a filegram:

1. The filegram **sender** creates a FILEGRAM-type template for a specified file (Create/Edit Filegram Template option). The Display Filegram Template option can be used to review the template.
2. The **sender** can optionally designate specifiers with the Specifiers option. These are used for matching existing entries with filegrams.
3. The **sender** then generates a filegram for a specific entry (Generate Filegram option). The filegram is placed into a MailMan message. The filegram sender sends this message to an individual or individuals at a remote or local destination.
4. The **recipient** receives the Filegram with MailMan, reading the mail message containing the filegram, and forwarding it to S.DIFG-SRV-

HISTORY. This is a special server that loads the message into the recipient's FILEGRAM HISTORY file (#1.12) and sets up the interface between Fileman and MailMan on the target system.

5. If you want to install the filegram on a system other than the one where you received it, instead of immediately forwarding to the S.DIFG-SRV-HISTORY server, forward the message to an individual on the ultimate target system (who in turn forwards it to *their* S.DIFG-SRV-HISTORY server.
6. Both the **sender** and the **recipient** can use the View Filegram option to inspect the filegram.
7. Then the **recipient** of the filegram on the target system uses the Install/Verify Filegram option to install the filegram into the destination file.
8. **Senders** and **recipients** can delete a filegram at any time.

The recipient can choose to modify the S.DIFG-SRV-HISTORY server or create another server to aid in the installation of filegrams.

REF: For additional information about setting up servers, see the Kernel and MailMan documentation.

Filegram Steps

Create/Edit Filegram Template Option

Use the Create/Edit Filegram Template option to create a FILEGRAM-type template or edit an existing FILEGRAM-type template. A FILEGRAM-type template is similar to a regular PRINT template without any formatting instructions. You always receive the "STORE FILEGRAM LOGIC IN TEMPLATE:" prompt, no matter how many fields you identify.

REF: For information about regular PRINT templates, see the "Print: How to Print Reports from Files" chapter of the *Fileman User Manual*.

The Create/Edit Filegram Template option is the first step in developing a filegram; there will not be a filegram without the template. This template is also used in the archiving process. Before using this option, you may wish to familiarize yourself with the file(s) and fields involved.

The following dialogue illustrates how to create a FILEGRAM-type template:

```
    OUTPUT FROM WHAT FILE: CHANGE
    FIRST SEND CHANGE FIELD: ??
```

You can enter **ALL** at this prompt, if you want to include all fields in the file in your filegram. **ALL** can also be used in existing file navigation paths. Enter **[?** at this prompt to get a listing of existing FILEGRAM-type templates for the selected file.

Here, two question marks ("**??**") requests a list of the fields in the file.

```
CHOOSE FROM:
    .01   NAME
    1     VERSION
    2     TAG
    3     ROUTINE
    4     CHANGE
    5     REPORTER (multiple)
    6     DATE CHANGED
    7     PROGRAMMER
    9     BUG OR FEATURE
   10     PURPOSE
   11     DESCRIPTION (word-processing)
```

In the example that follows, the PROGRAMMER field is a pointer to the NEW PERSON file (#200).

```
FIRST SEND CHANGE FIELD: 1 <Enter>  VERSION
 THEN SEND CHANGE FIELD: 3 <Enter>  ROUTINE
 THEN SEND CHANGE FIELD: 4 <Enter>  CHANGE
 THEN SEND CHANGE FIELD: 5 <Enter>  REPORTER (multiple)
  FIRST SEND REPORTER SUB-FIELD: .01
   THEN SEND REPORTER SUB-FIELD: <Enter>
 THEN SEND CHANGE FIELD: 6 <Enter>  DATE CHANGED
 THEN SEND CHANGE FIELD: 7 <Enter>  PROGRAMMER
 THEN SEND CHANGE FIELD: 11 <Enter>  DESCRIPTION (word-
processing)
 THEN SEND CHANGE FIELD: <Enter>
STORE FILEGRAM LOGIC IN TEMPLATE: ZZTEST FILEGRAM
  Are you adding 'ZZTEST FILEGRAM' as a new PRINT TEMPLATE?
No// Y
```

The template for your filegram is now set up. Edit this template just like you would any other PRINT template.

NOTE: You do not have to include the .01 field, because it is automatically included for use as a lookup value.

To send logical records in a filegram, simply use file navigation to and from existing files at the "SEND FIELD:" prompts.

REF: For a discussion of relational navigation using forward and backward pointers, see the "Relational Navigation" section.

Display Filegram Template Option

The Display Filegram Template option displays the FILEGRAM-type template in a two-column format (like the Inquire to File Entries option). Multiple-type fields are shown last in the display, no matter what their field number.

Here is an example of the output produced by the Display Filegram Template option:

```
Select FILEGRAM OPTION: DISPLAY FILEGRAM TEMPLATE
SELECT FILEGRAM TEMPLATE: ZZTEST FILEGRAM


NAME: ZZTEST FILEGRAM                 DATE CREATED: AUG 24, 1989
  READ ACCESS:  @           FILE: 1001
  USER #:  29               WRITE ACCESS:  @
  DATE LAST USED: AUG 24, 1989
ORDER:  1                   FILEGRAM FILE:  1001
 LEVEL:  1                        DATE LAST STORED: AUG 24, 1989
FIELD ORDER:  1             FIELD NUMBER:  .01
```

```
CAPTION (c):  NAME
FIELD ORDER:  2            FIELD NUMBER:  1
CAPTION (c):  VERSION
FIELD ORDER:  3            FIELD NUMBER:  3
CAPTION (c):  ROUTINE
FIELD ORDER:  4            FIELD NUMBER:  4
CAPTION (c):  CHANGE
FIELD ORDER:  5            FIELD NUMBER:  6
CAPTION (c):  DATE CHANGED
FIELD ORDER:  6            FIELD NUMBER:  7
CAPTION (c):  PROGRAMMER
FIELD ORDER:  7            FIELD NUMBER:  11
CAPTION (c):  DESCRIPTION
ORDER:  2                  FILEGRAM FILE:  1001.05
  LEVEL: 2                 PARENT:  1001
  CROSS-REFERENCE: MULTIPLE   USER RESPONSE TO GET HERE:
REPORTER
  DATE LAST STORED: AUG 24, 1989
FIELD ORDER:  1                  FIELD NUMBER:  .01
CAPTION (c):  REPORTER

FIRST PRINT FIELD:  S DIFGT=315 D FG^DIFGB;X//
COMPILED (c):  N
```

The code in the FIRST PRINT FIELD has special meaning to Fileman.

Specifiers Option

The filegram sender uses the Specifiers option to identify a particular field in the file as a reference point to use when *installing* the filegram. The value of this field in the filegram is compared to values in the entries at the target site. The values *must* match for a filegram to be installed. If the specifier has a unique value for every entry in the file and is cross-referenced, that cross-reference is used to locate an entry. This reduces the search time and increases accuracy. Specifiers can be compared to identifiers: unlike identifiers, which are used for user interaction purposes, specifiers are used for transaction purposes. Specifiers are optional.

The dialogue that follows creates a specifier in a sample PATIENT file:

```
Select FILEGRAM OPTION: 5 <Enter>  SPECIFIERS
```

```
OUTPUT FROM WHAT FILE: PATIENT

Select FIELD: 3 <Enter>  SSN

Want to make SSN a specifier?  NO// YES

Is the value of this field unique for each entry?  NO// YES
```

Answer YES only if you have a regular cross-reference on the field that you are making a specifier. A field can be a specifier without being unique.

If you answer YES, a dialogue similar to the following occurs:

```
Select one of the following:

  1 C  REGULAR

If one of the above provides a direct look-up by SSN, please
enter its number or name: 1
```

To delete a specifier:

```
Select FIELD: 3 <Enter>  SSN

SSN is already a specifier.
Do you want to delete it?  NO// YES
```

Generate Filegram Option

The Generate Filegram option is used by a filegram sender. Be sure that the DUZ correctly identifies the filegram sender; the DUZ is used to identify the filegram's sender to the recipient. The option creates a filegram in MailMan message format after you designate a file, FILEGRAM-type template, and a file entry. Concurrently, it creates a record in the FILEGRAM HISTORY File (the FILEGRAM HISTORY file [#1.12] points to the MESSAGE file [#3.9].) The record created in the FILEGRAM HISTORY file is called a filegram history. The filegram history allows Fileman to differentiate between a filegram message and a mail message. After the filegram is placed into the mail message, you can send it to an individual at any established address.

You can only send one entry at a time. The following dialogue illustrates the generation of a filegram:

```
Select FILEGRAM OPTION: GENERATE FILEGRAM
OUTPUT FROM WHAT FILE: CHANGE
Select FILEGRAM TEMPLATE: ZZTEST FILEGRAM
Select CHANGE NO.: 334
Send mail to: SYSTEM,MGR@METRODB.VA.GOV
And send to: <Enter>
```

Receiving Filegrams with MailMan

Filegram messages do not appear as NEW mail at the receiving site. After the mail message is received, the filegram recipient *must* read the mail message. Then, it should be forwarded to S.DIFG-SRV-HISTORY on the target system (i.e., the system on which the filegram will be installed). This is a special server used to load the message into the recipient's FILEGRAM HISTORY file (#1.12) and to set up the interface between Fileman and MailMan at the target system.

Here is an example of what the recipient would see when reading and forwarding a filegram:

```
Subj: FILEGRAM for entry #334 in CHANGE FILE (1001).  [#186309]
25 Aug 89  10:00  20 lines
From: SITE,MANAGER  in 'IN' basket  Page 1
--------------------------------------------------------------
$DAT^CHANGE^1001^N^
CHANGE^1001^L=334
  BEGIN:CHANGE^1001@1
    SPECIFIER:VERSION^1=17.4
  END:CHANGE^1001
  VERSION^1=17.4
  ROUTINE^3=DICATT5
  CHANGE^4=Changed DIED to DIE0
  DATE CHANGED^6=APR 13, 1987
  PROGRAMMER^7=FMPROGRAMMER
  DESCRIPTION^11=wp
"This change enables incredibly wonderful things"
"to occur."
.
  REPORTER^5^L=FMEMPLOYEE,10
```

```
    BEGIN:REPORTER^1001.05@2
    END:REPORTER^1001.05
    REPORTER^.01=FMEMPLOYEE,10
  ^
$END DAT

Enter message action (in IN basket): IGNORE// **F**
Forward mail to: **S.DIFG-SRV-HISTORY**

  SENDING A MESSAGE
```

View Filegram Option

Entries are made into the FILEGRAM HISTORY file (#1.12) at the sending site when the filegram is generated and at the receiving site by the S.DIFG-SRV-HISTORY server. The View Filegram option allows the filegram sender or recipient to inspect the filegram. Select this option and answer the "Select FILEGRAM HISTORY:" prompt with a question mark to get a listing of available filegram histories. You can see the filegram history just created by using the **<Spacebar><Enter>** or another filegram history by entering its internal entry number or date/time.

The following example shows you what a simple (without pointers) filegram looks like. The View Filegram option will show this information to a filegram's sender. The receiver sees additional information about the transmission of the mail message including the network mail path taken to the target system.

```
FILEGRAM for entry #334 in CHANGE FILE (#1001).

Sent on 25 AUG 1989 @ 09:00 by FMUSER,FOUR

$DAT^CHANGE^1001^N^
CHANGE^1001^L=334
  BEGIN:CHANGE^1001@1
    SPECIFIER:VERSION^1=17.4
  END:CHANGE^1001
  VERSION^1=17.4
  ROUTINE^3=DICATT5
  CHANGE^4=Changed DIED to DIE0
  DATE CHANGED^6=APR 13, 1987
  DESCRIPTION^11=wp
```

```
"This change enables incredibly wonderful things"
"to occur."
.
  REPORTER^5^L=FMPATIENT,10
    BEGIN:REPORTER^1001.05@2
    END:REPORTER^1001.05
    REPORTER^.01=FMPATIENT,10
  ^
$END DAT
```

Install/Verify Filegram Option

The filegram recipient uses the Install/Verify Filegram option to install a filegram from a MailMan message into a file on the target system. Choose a filegram history by entering its date/time or by using a **<Spacebar><Enter>** (for the last used history) at the "Select FILEGRAM HISTORY:" prompt. The destination file for the filegram entry *must* be in place on the target system for a successful installation!

CAUTION: The installation will more likely succeed if the destination file is a replica of the sending file.

The message "DONE" is displayed if the install was successful. If not successful, an UNSUCCESSFUL INSTALLATION message is returned with an error code.

REF: For a list of error codes identifying their meaning, see the "^DIFG" section in the "Filegrams API" chapter in the "Other APIs" section in the *Fileman APIs Manual*.

Deleting a Filegram

Delete a filegram by removing the filegram's entry from the FILEGRAM HISTORY file (#1.12). As shown in the following dialogue, use the Enter or Edit File Entries option. Enter the at-sign ("@") at the DATE/TIME prompt of the filegram history you want to delete.

```
Select OPTION: ENTER OR EDIT FILE ENTRIES
```

```
INPUT TO WHAT FILE: **FILEGRAM HISTORY**
EDIT WHICH FIELD:  ALL// **<Enter>**

Select FILEGRAM HISTORY: **8-24-1994@11:10:00**
DATE/TIME:  AUG 24, 1994@11:10:00// **@**
  SURE YOU WANT TO DELETE THE ENTIRE FILEGRAM HISTORY? **YES**
```

Chapter 14: Archiving

NOTE: To use Fileman's archiving procedure, Kernel 6.5 or later *must* be installed on your system. To use the Find Archived Records option to retrieve records archived with Fileman versions earlier than 20.0, Kernel 7.1 or later *must* be installed on your system.

In general computer terms, archiving is a procedure that permits you to remove data from an online database and place that data into a low-cost storage medium, such as magnetic tape, for long-term retention.

Archiving in Fileman is a complete purge that simply clears data from your system. In other words, no data restoration is provided! However, there is a data retrieval option.

This facility is a prototype and supported only as a developer's tool. You *must* access your archiving options directly through Fileman from programmer mode (e.g., >D Q^DI).

Fileman performs the archiving function by searching through file entries using specified criteria, extracting and transporting the selected entries by filegrams to temporary storage in the ARCHIVAL ACTIVITY file (#1.11), and then simply writing the data to permanent storage. The FILEGRAM-type template used to transport an archive activity can be created during the archiving session (in the Archiving menu options), or an existing FILEGRAM-type template created using the Filegram options can be used.

Considerations Before Archiving

The following summarizes some of the important items to note regarding the archiving facility. Consider them before you begin archiving:

- We strongly encourage you to have a current backup of your files before archiving.

- Archiving is not user-specific. In other words, archiving is attached to a file not a user. For your own protection, please be aware that someone other than yourself can complete or change an existing archiving activity.
- Data from logical and physical files can be archived, but only the data from the physical (primary) file can be purged (removed).
- Fileman *must* be able to collect and print the data (using the search criteria) before you can create an archiving activity. That is, the Select Entries to Archive option *must* be run to completion such that the selected entries are printed (and can thus be stored in the ^DIBT SORT TEMPLATE global).
- If you plan to keep a hard copy of the printed entries for future reference, design your PRINT template to facilitate review. You do not need to print all of the fields' values that will be archived, but you may want to include those that, in combination, will uniquely identify the entry. Save the PRINT template for future use when archiving.
- When the archived entries are written to temporary storage, the corresponding original entry disappears from the user's view. (A new node, subscripted with -9, is added to the original entry so that it will be bypassed by the usual Fileman calls.)
- If data to be archived is contained in a Multiple field, then the entire entry *must* be archived. You cannot archive a subentry by itself.
- A FILEGRAM-type template is the only template type allowed in the Write Entries to Temporary Storage option.
- You cannot have more than one archiving activity on a file at a time.
- Data selected for archiving can be permanently saved to any **sequential** storage media, for example: SDP, a VMS file, magnetic tape, or a removable disk pack.
- Depending on the number of entries involved, be prepared for the search (the Select Entries to Archive option) and write (the Write Entries to Temporary Storage option) processes to be time-consuming.
- A brief history of who performed the various archive steps and when they were accomplished is saved in the ARCHIVAL ACTIVITY file.
- You can cancel your archiving process, by using the Cancel Archival

Selection option, at any time before the Purge Stored Entries option is used.

- The Find Archived Entries option can be used to verify that the archive medium contains all the information intended to be archived.
- The Purge Stored Entries option completely deletes data from the file being archived and from the ARCHIVAL ACTIVITY file.
- You cannot purge archived entries until you have moved selected entries to permanent storage. Thus, you need not worry about losing entries before they are archived.
- You cannot start a second archive from a file until you purge or cancel the existing archiving activity on that file.

Archiving Process, including Archiving Options (1-9)

The order of the options on the Archiving submenu reflects the sequence of steps in which you ordinarily do archiving. Access the Archiving submenu from the Other Options submenu:

```
Select OPTION: OTHER OPTIONS
Select OTHER OPTION: ARCHIVING

Select ARCHIVE OPTION: ?
 ANSWER WITH ARCHIVE OPTION NUMBER, OR NAME
CHOOSE FROM:
   1              SELECT ENTRIES TO ARCHIVE
   2              ADD/DELETE SELECTED ENTRIES
   3              PRINT SELECTED ENTRIES
   4              CREATE FILEGRAM ARCHIVING TEMPLATE
   5              WRITE ENTRIES TO TEMPORARY STORAGE
   6              MOVE ARCHIVED DATA TO PERMANENT STORAGE
   7              PURGE STORED ENTRIES
   8              CANCEL ARCHIVAL SELECTION
   9              FIND ARCHIVED ENTRIES
```

Select Entries to Archive

The Select Entries to Archive option creates the archiving activity. It is used similarly to the Search File Entries option. The Select Entries to Archive option is the first step in developing an archiving activity and is very important since there cannot be any archiving without the SEARCH template created in this option.

TIP: It's important to know which entries you want archived and where you want them stored before you start the archiving process!

This mandatory archiving option really performs three important functions:

1. A search for file entries that meet your specified search criteria or condition (truth test) occurs first. The results of this search are then stored in a template you specify. You *must* store these results in a template!
   REF: For guidance on how to use the search option procedures, see the "Search" chapter in the *Fileman User Manual*.
2. After storing the results of the search in a template, a sort of the file by any field *must* occur. This sort is as important as the search portion of this option! Do not use any sort criteria that contains M code. Be careful, if you limit the sort range, you will limit the entries selected for archiving. You can delete unwanted entries later by using the Add/Delete Selected Entries option.
3. Finally, a print of the fields to be archived to a printer or the screen (CRT) *must* occur. Printing fields that uniquely identify the entries being archived gives you a permanent record of the archived entries.

You *cannot* create an archiving activity until *at least one* entry is located according to your search criteria, sorted, and printed to a printer or a terminal!

NOTE: If a subentry to be archived is contained in a Multiple field, the entire entry *must* be archived.

An example of the dialogue you may encounter follows. Notice the sequence of search, sort, and print:

```
ARCHIVE FROM WHAT FILE: CHANGE

  -A- SEARCH FOR CHANGE FIELD: .01 <Enter>  NO.
  -A- CONDITION: LESS THAN
  -A- LESS THAN: 900

  -B- SEARCH FOR CHANGE FIELD: <Enter>

  IF:  A// <Enter>  NO. LESS THAN 900

STORE RESULTS OF SEARCH IN TEMPLATE: ZZTEST TEMPLATE
 Are you adding 'ZZTEST TEMPLATE' as a new SORT TEMPLATE?  No//
Y <Enter>  (Yes)

SORT BY: VERSION
START WITH VERSION:  FIRST// <Enter>
  WITHIN VERSION, SORT BY: <Enter>

FIRST PRINT FIELD: .01 <Enter>  NO.
THEN PRINT FIELD: VERSION
THEN PRINT FIELD: PROGRAMMER
THEN PRINT FIELD: <Enter>
HEADING:  CHANGE ARCHIVE SEARCH  Replace <Enter>
DEVICE: <Enter>


CHANGE ARCHIVE SEARCH              AUG 30, 1992  10:59   PAGE 1
   NO.          VERSION           PROGRAMMER
----------------------------------------------------------------

  101          17.10              FMPROGRAMMER,25
  102          17.32              FMPROGRAMMER,26
  103          17.35              FMPROGRAMMER,26
                      3 MATCHES FOUND.
```

After using this option, the status of your archiving activity is SELECTED. The status of an archiving activity can be SELECTED, EDITED, ARCHIVED (TEMPORARY), ARCHIVED (PERMANENT), or PURGED. You may identify a file at the "ARCHIVE FROM WHAT FILE:" prompt and get the response that follows:

```
    There is already an outstanding archiving activity.
    Please finish it or CANCEL it.
```

This message means that the file you identified has already been selected for archiving and that archiving activity has not been completed; a second one cannot be started yet. Since the ARCHIVAL ACTIVITY file is being shared by both archiving and extract activities, the italicized word in the message will say *extract* if the outstanding activity on the file identified is an extract.

Add/Delete Selected Entries

Use the Add/Delete Selected Entries option to add entries to or delete them from the archiving activity. This is an easy way to clear out unwanted entries or add needed ones before archiving.

This option uses the Inquire to File Entries option to display the selected entries and then allows you to add or delete an entry. If you add or delete an entry to an established archiving activity, then the status of the activity will change to EDITED.

The Add/Delete Selected Entries option will *not* allow you to edit an archiving activity list after the Write Entries to Temporary Storage option has been done. If you need to change the archiving activity list after writing to temporary storage, you *must* cancel that archiving activity and start a new one.

When using this option, you first select the archiving activity number that you want to modify. You can also identify the archiving activity by its file number or file name. Then, choose the entry that you want to add or delete.

Here is an example in which an entry is being added to the archiving activity:

```
Select ARCHIVE OPTION: ADD/DELETE <Enter>  SELECTED ENTRIES
Select ARCHIVAL ACTIVITY: ?
  ANSWER WITH ARCHIVAL ACTIVITY ARCHIVE NUMBER, OR FILE
CHOOSE FROM:
   1        Fileman CHANGE          08-05-89
ARCHIVED(PERMANENT)     SELECTOR: FMUSER,FIVE     ARCHIVING
   3        CHANGE                     08-30-92     SELECTED
SELECTOR: FMUSER,SIX     ARCHIVING

Select ARCHIVAL ACTIVITY: 3

Select CHANGE NO.: 330
NO.:  330                           VERSION:  17.09
 PROGRAMMER:  FMPROGRAMMER,27        ROUTINE:  DIL2
 DATE CHANGED:  OCT 24, 1986

ADD this entry TO the ARCHIVAL SELECTION?  YES// <Enter>
```

If you enter two question marks ("**??**") at the prompt where you select the entry for addition or deletion, a list of the file's entries is displayed. Those that are already part of the archiving activity are identified by "*ON ARCHIVE LIST*". The "ADD this entry TO the ARCHIVAL SELECTION? YES//" question appears if you have selected an entry *not* already on the list for archiving. A "DELETE ..." question appears if you have selected an entry already on the list.

Print Selected Entries

The Print Selected Entries option displays each entry from a selected archiving activity in a regular print or a filegram format (depending on the template you identify). The archiving activity entries are printed on whatever device you indicate. You can use this option for an archiving activity with any status except PURGED.

The example below illustrates the regular print format:

```
Select ARCHIVE OPTION: PRINT <Enter>  SELECTED ENTRIES
Select ARCHIVAL ACTIVITY: 3 <Enter>       CHANGE  08-30-92
EDITED     SELECTOR:FMPATIENT,2     ARCHIVING
Enter regular Print Template name or fields you wish to see
printed
on this report of entries to be archived.

FIRST PRINT FIELD: [ZZTEST TEMPLATE
```

```
CHANGE ARCHIVAL ACTIVITY                AUG 30, 1992  11:09
PAGE 1
NO.             VERSION                 PROGRAMMER
----------------------------------------------------------------

101             17.10                   FMPROGRAMMER,25
102             17.32                   FMPROGRAMMER,26
103             17.35                   FMPROGRAMMER,26
330             17.09                   FMPROGRAMMER,30
```

Shown below is an example of a Filegram format produced by naming a FILEGRAM-type template to print (only a single entry is shown.):

```
FIRST PRINT FIELD: [ZZTESTFILEGRAM

CHANGE ARCHIVAL ACTIVITY            AUG 30, 1992  11:09   PAGE 1
----------------------------------------------------------------

          NO.:  330
$DAT^CHANGE^16000^N^
CHANGE^16000^L=330
 BEGIN:CHANGE^16000@1
   SPECIFIER:VERSION^1=17.09
   IDENTIFIER:PROGRAMMER^7=FMPROGRAMMER,30
 END: CHANGE^16000
 NO.^.01=330
 VERSION^1=17.09
$END DAT
```

Create Filegram Archiving Template

The Create Filegram Archiving Template option creates a FILEGRAM-type template, which is the template used in the Write Entries to Temporary Storage option. A filegram is the tool used for extracting and transporting archived data to temporary storage.

REF: For further explanation, see the "Write Entries to Temporary Storage" section.

You can create a new FILEGRAM-type template with this option or edit an existing one created using the Filegram or Archiving options.

NOTE: Only those fields defined in the FILEGRAM-type template, the .01 field, and any PRIMARY KEY or Identifier fields will be archived to permanent storage.

The PRIMARY KEY is available as of Version 22.0.

You create a FILEGRAM-type template just as you create a PRINT template, except only valid field numbers or names can be entered. You cannot include print qualifiers. You always get the "STORE FILEGRAM LOGIC IN TEMPLATE:" prompt, no matter how many fields you identify.

FILEGRAM-type templates are stored in the PRINT TEMPLATE file (#.4), so the FILEGRAM-type template is referred to as a PRINT template in the dialogue. However, Fileman can distinguish between PRINT and FILEGRAM-type templates.

If you want to edit an existing FILEGRAM-type template, identify it at the "FIRST SEND FIELD:" prompt by entering "**[Filegram templatename**". Otherwise, specify the fields you want included in the archive. (The .01 field and the file's PRIMARY KEY and Identifier fields will always be sent.) In the example that follows, all fields are being sent.

NOTE: The PRIMARY KEY is available as of Version 22.0.

Here is an example of creating a FILEGRAM-type template:

```
Select ARCHIVE OPTION: CREATE FILEGRAM ARCHIVING TEMPLATE

OUTPUT FROM WHAT FILE: CHANGE

FIRST SEND CHANGE FIELD: ALL
Do you mean ALL the fields in the file? No// Y
 THEN SEND CHANGE FIELD: <Enter>

STORE ARCHIVE LOGIC IN TEMPLATE: CHANGE FILEGRAM
  Are you adding 'CHANGE FILEGRAM' as a new PRINT TEMPLATE?
No// YES
```

Write Entries to Temporary Storage

The Write Entries to Temporary Storage option writes your selected archiving activities to the ARCHIVAL ACTIVITY file into a WORD-PROCESSING field. This step is in preparation for moving the data to permanent storage. You *cannot* archive to permanent storage unless you use the Write Entries to Temporary Storage option first. Needless to say, this file could grow quite large and will shrink only after the Purge Stored Entries option has been run.

After using this option, the archived entries will appear to be missing from the primary file. This protective measure assures selected entries *cannot* be edited so that entries in the file will match the archived version. The entries are *not* really gone, merely locked.

Also, after using this option, you *cannot* add or delete entries to an archiving activity. If changes in the selection of entries for archiving is necessary, you have to cancel this activity and restart.

⚠ CAUTION: This process can be quite time-consuming! It can be queued at the "DEVICE:" prompt. After it is completed, the status of your archiving activity is ARCHIVED (TEMPORARY).

In the dialogue below, the entries in Archiving Activity #3 are being sent to temporary storage using the ZZTESTFILEGRAM FILEGRAM-type template. The task is being queued with output sent to PRINTER. The resulting report contains a header, dots, and archiving totals, as shown below:

```
Select ARCHIVE OPTION: WRITE <Enter>  ENTRIES TO TEMPORARY
STORAGE
Select ARCHIVAL ACTIVITY: 3 <Enter>       CHANGE  08-30-92
EDITED     SELECTOR:FMEMPLOYEE,J     ARCHIVING

You MUST enter a FILEGRAM template name.  This FILEGRAM template
will be used to actually build the archive message.

PRINT TEMPLATE: ZZTESTFILEGRAM <Enter>  **FILEGRAM**  (AUG 30,
1992)   USER
#60 FILE #16000

DEVICE: Q <Enter>  UEUE TO PRINT ON
DEVICE: PRINTER

CHANGE ARCHIVING ACTIVITY          AUG 30, 1992 15:01  PAGE 1
---------------------------------------------------------------

....
4 ITEMS HAVE BEEN ARCHIVED
```

Move Archived Data to Permanent Storage

Once you have written an archiving activity to temporary storage, you can use the Move Archived Data to Permanent Storage option. If you choose an archiving activity that has not yet been written to temporary storage, a warning is issued.

This option will do several things not necessarily in the order listed here.

1. It will print an Archive Activity report, which prints information from the ARCHIVAL ACTIVITY file (#1.11), such as the archiver (i.e., the person who selected this option), the archival activity number, an index of all archived entries, and the search criteria used during the Select Entries to Archive process, etc.
2. It will build an index of all archived entries and write this index at the beginning of the archived file. An item in the index contains the .01 value of the entry, along with all PRIMARY KEY and Identifier values for that entry.
   NOTE: The PRIMARY KEY is available as of Version 22.0.
3. It will prompt for Archive Device Label information. This information will usually represent some naming convention that Systems Managers use as physical label for devices such as tape. If using a disk file, the full disk file name will be presented as a default. This device label information along with the index printed on the Archive Activity report should be useful in locating an archived entry and the device to which it was archived.
4. Lastly, this option will move archive data to permanent storage.

Permanent storage is considered any sequential storage media like SDP, VMS file, magnetic tape, or disk data set.

You can send to more than one permanent storage location without having to recreate the archiving activity. When the archived data is moved to permanent storage, every line contained in the temporary storage word processing field is simply read and then written to a sequential medium.

The status of your archiving activity after using this option is ARCHIVED (PERMANENT).

In the example below, the archiving activity is identified by file name, "CHANGE." It is going to be archived to the specified tape:

```
Select ARCHIVE OPTION: MOVE ARCHIVED DATA TO PERMANENT STORAGE
Select ARCHIVAL ACTIVITY: CHANGE <Enter>       3  CHANGE  08-30-
92     ARCHIVED(TEMPORARY)     SELECTOR:FMEMPLOYEE,J
ARCHIVING

NOTE:  This option will 1) print an archive activity report to
specified PRINTER DEVICE and 2) will move archive data to
permanent
storage to specified ARCHIVE STORAGE DEVICE.

Select some type of SEQUENTIAL storage media, such as SDP, TAPE,
or
DISK FILE (HFS) for archival storage.

PRINTER DEVICE: PRINTER

ARCHIVE STORAGE DEVICE: MAGTAPE Parameter ("CAVL":0:2048)
DO YOU WANT YOUR OUTPUT QUEUED?  NO// YES
Requested Start Time:  NOW// <Enter>

ARCHIVE DEVICE LABEL:  ISC6V3$MUA0:ARCHIVE;083092;3// <Enter>
```

If the archive storage device is tape, an archive device label is generated for you, as depicted in the previous example. You can override this label by entering your own device label information.

If you select an archive storage device that is non-sequential, a warning is issued. You are then given the option to continue the archiving process.

CAUTION: If you specify a device that can be overwritten (e.g., SDP) or that does *not* electronically store data (e.g., a printer), the data will be *forever* irretrievable.

Depending on what devices are selected, and whether queueing was requested for either device, different warnings will be issued informing the user of either steps he need to take or steps that the program will take as a result of the queueing request.

Here is an example of an archive activity report:

```
ARCHIVE ACTIVITY REPORT                    AUG 30,1992  PAGE: 1
---------------------------------------------------------------
ARCHIVAL ACTIVITY: 3
ARCHIVE DEVICE LABEL INFORMATION:  ISC6V3$MUA0:ARCHIVE;083092;3
PRIMARY ARCHIVED FILE: CHANGE (#16000)
ARCHIVER: FMEMPLOYEE,J
SEARCH CRITERIA:
     .01 LESS THAN 900


INDEX INFORMATION:

NO.        VERSION          PROGRAMMER
101        17.10            FMPROGRAMMER,25
102        17.32            FMPROGRAMMER,26
103        17.35            FMPROGRAMMER,26
330        17.09            FMPROGRAMMER,30

*** PLEASE KEEP THIS FOR FUTURE REFERENCE ***
```

Purge Stored Entries

Before running the Purge Stored Entries option, use the Find Archived Entries option to verify that the archive medium contains the complete archived data for an archiving activity by searching for the last record listed on the index.

This option is used to remove archived data from both the archived file (our example is the CHANGE file) and the ARCHIVAL ACTIVITY file. A brief history of who performed the various archiving steps and when is saved in the ARCHIVAL ACTIVITY file.

If you select an archiving activity for purging that has not been sent to the archives, then you will receive the message:

```
    Data has NOT YET been archived to PERMANENT storage!
```

You will see the following dialogue when purging permanently archived data:

```
Select ARCHIVE OPTION: PURGE STORED ENTRIES

BEFORE YOU PURGE, MAKE SURE THAT YOUR ARCHIVE MEDIUM IS
READABLE!
YOU MAY USE THE FIND ARCHIVED ENTRIES OPTION TO FIND THE LAST
ARCHIVED RECORD APPEARING ON THE INDEX.

Do you want to proceed?? NO// YES
Select ARCHIVAL ACTIVITY:  3 <Enter>       CHANGE  08-30-92
ARCHIVED(PERMANENT)
SELECTOR:FMEMPLOYEE,J     ARCHIVING

This option will DELETE DATA from both CHANGE
and from the ARCHIVAL ACTIVITY file.

Are you sure you want to continue? NO// YES
```

By answering YES to the "Are you sure you want to continue? NO//" prompt, the entries will be ***immediately*** deleted! Be sure this is your decision.

```
The entries will be deleted in INTERNAL NUMBER order.

<< 4 ENTRIES PURGED >>
```

Cancel Archival Selection

The Cancel Archival Selection option is used to cancel an archiving activity before the Purge Stored Entries option occurs. You are warned if the archiving activity has already been moved to permanent storage.

When you cancel an archiving activity, the entry in the ARCHIVAL ACTIVITY file is deleted. Also, entries that were locked after being moved to temporary storage can again be read and edited.

See the following dialogue for canceling an archival activity:

```
Select ARCHIVE OPTION: CANCEL ARCHIVAL SELECTION
Select ARCHIVAL ACTIVITY: CHANGE <Enter>       3 CHANGE  08-30-
92     EDITED
SELECTOR:FMEMPLOYEE,J     ARCHIVING

Are you sure you want to CANCEL this ARCHIVING ACTIVITY?  NO//
YES

>>> DONE <<<
```

Find Archived Entries

The Find Archived Entries option scans an archive file and retrieves an archive entry (or entries) that matches the NAME field (#.01) and/or the PRIMARY KEY and/or Identifier field(s) of the requested entry. On the prompts for lookup values and on the resulting report, PRIMARY KEY and Identifier fields are both called "Identifiers." The information contained in the archived entry is printed in CAPTIONED format [field name (field number) = value]. This option should only be used with tapes or files archived under Fileman 19.0 or later.

NOTE: The PRIMARY KEY is available as of Fileman 22.0.

Several requests for archived entries can be made at a time. An "(id)" next to a prompt indicates an Identifier, PRIMARY KEY, or Specifier field. One set of all the prompts makes one request.

The matching process is dependent on the presence or absence of an index on the archive file or tape. The matching process for archive files or tapes with no index can be more time-consuming, since it has to read the entire archived file to determine all matches. The matching process finds a match when the values of all the answered prompts match with that of an archived entry. If a partial value is typed at any prompt, the matching process finds all matches that begin with the partial value for that particular prompt.

See the following dialogue for finding archived entries:

```
Select ARCHIVE OPTION: FIND ARCHIVED ENTRIES

This option will scan your archived file and will attempt to
retrieve entries that match the name (.01) field and/or the
identifier field(s) of the archived file.

Magnetic tapes should be opened with variable length records.

SEQUENTIAL ARCHIVE DEVICE: HFS DISK FILE
HOST FILE NAME: TMP.TMP// ARC.DAT

You are reading archived information from the CHANGE file.
Do you want to continue? YES// <Enter>

Multiple requests may be made.
One set of all prompts makes one request.

This archived file contains an index of all archived entries.
Do you want to see the index now? YES// <Enter>

NO.          VERSION          PROGRAMMER

101          17.10            FMPROGRAMMER,25
102          17.32            FMPROGRAMMER,26
103          17.35            FMPROGRAMMER,26
330          17.09            FMPROGRAMMER,30

Enter NO.: <Enter>
Enter VERSION (id) : <Enter>
Enter PROGRAMMER (id) : FMPATIENT,30

Enter NO.: <Enter>
Enter VERSION (id) : <Enter>
Enter PROGRAMMER (id) : <Enter>

Enter NO.: <Enter>
```

```
PRINT FOUND ENTRIES TO DEVICE: <Enter>  DECSERVER

Searching archived file...
Formatting found entries...
Press RETURN to continue or '^' to exit: <Enter>
ARCHIVE RETRIEVAL LIST                                   AUG 30,1992
PAGE: 1
REQUEST: 1
  PROGRAMMER = FMPROGRAMMER,2
---------------------------------------------------------------

ARCHIVE FILE: CHANGE (#16000)

  LOOKUP VALUE (#.01): 330
    IDENTIFIERS:
      VERSION (#1) = 17.09
      PROGRAMMER (#7) = FMPROGARMMER,30
    FIELDS:
      FIELD NAME: NO. (#.01) = 330
      FIELD NAME: VERSION (#1) = 17.09

           MATCHES FOUND: 1
```

ARCHIVAL ACTIVITY File

The status of an archiving activity is displayed in the listing of activities when a question mark is entered at most of the Archiving options. You can also use the Inquire to File Entries option on the ARCHIVAL ACTIVITY file (#1.11) to obtain information about past or pending archiving activities. The amount of information you receive depends on the status of the archiving activity.

In summary, the information contained in this file describes who archived, status of the archive, when entries were archived, the number of items archived, name of the file archived, archiving activity internal entry number, and the SEARCH and PRINT templates used in archiving. If an archiving activity is purged, the ARCHIVAL ACTIVITY file contains the purge and archive date and the person who did the purge.

Beginning with Fileman 20.0, the ARCHIVAL ACTIVITY file contains data about both archiving and extract activities. A file can have only *one* active activity at a time—either an archiving activity or an extract activity.

Archiving activities are selectable only within the Archiving options. When you are doing an inquire on the ARCHIVAL ACTIVITY file, the word ARCHIVING will appear for all archiving activities.

Appendix A: Advanced Edit Techniques

Field Value Stuffing

You can make the editing process quicker, easier, and more accurate by "stuffing" field values, when appropriate. The amount of data that needs to be entered from the keyboard can be reduced by providing responses that can be verified by pressing the **Enter** key or that are automatically put into the file.

Unvalidated Stuffs: (4////)

If you have programmer access, you can define a default that does not go through the INPUT transform by using four slashes ("////"). If you use this kind of default, you *must* show the internally stored value of the field.

For example, the SEX field has a DATA TYPE field value of SET OF CODES, where "m" stands for MALE and "f" stands for FEMALE; you could define a four-slash stuff like this:

```
    EDIT WHICH FIELD:  SEX////m
```

Variable Stuffs

An even more powerful kind of default is the variable default. In this mode, you specify, not a literal value like the word MALE, but rather a field name from which to calculate the default value for each entry being edited.

One example of the usefulness of this kind of default is a case where you are editing two fields that usually have the same value. Suppose that, for a set of patients, you want to enter a NEXT OF KIN field, followed by a BENEFICIARY field. Once you have typed a patient's NEXT OF KIN, you want to see that particular answer as the default value of BENEFICIARY.

The process would look like this:

```
INPUT TO WHAT FILE: **PATIENT**
EDIT WHICH FIELD: ALL// **NEXT OF KIN**
THEN EDIT FIELD: **BENEFICIARY//NEXT OF KIN**
  DO YOU MEAN 'NEXT OF KIN' AS A VARIABLE? YES// **<Enter>**
THEN EDIT FIELD: **<Enter>**

Select PATIENT NAME: **FMPATIENT,11**
NEXT OF KIN: **MRS CLOSERELATIVE FMPATIENT**
BENEFICIARY: MRS CLOSERELATIVE FMPATIENT// **<Enter>**

Select PATIENT NAME: **FMPATIENT,14**
NEXT OF KIN: **MR CLOSERELATIVE FMPATIENT**
BENEFICIARY: MR CLOSERELATIVE FMPATIENT// **MISS CLOSERELATIVE_2
FMPATIENT**
```

Here, Mrs. Sarah FMPATIENT ends up as both the NEXT OF KIN and BENEFICIARY for 11 FMPATIENT, while 14 FMPATIENT's NEXT OF KIN and BENEFICIARY are two distinct people.

A variable default value can be any computed expression—such as LAST VISIT DATE+365.

REF: For more information on computed expressions, see the "Computed Expressions" section.

WORD-PROCESSING Field Stuffing

The effect of stuffing values in a DATA TYPE field of WORD-PROCESSING is similar to defaults for other fields: the default value becomes the first line of the word-processing text. Also, you can stuff many lines of text into a DATA TYPE field of WORD-PROCESSING by use of a computed expression that has a Multiline value (e.g., another WORD-PROCESSING-type field).

Alternatively, you can automatically append data to a DATA TYPE field of WORD-PROCESSING by following the "//" or "///" with a "+" sign. This means add on the following text to whatever may already be on file. Let's take the example of our WORD-PROCESSING-type HISTORY field data in the PATIENT file (#200):

```
    EDIT WHICH FIELD: DIAGNOSIS
    EDIT WHICH DIAGNOSIS SUB-FIELD:  HISTORY//+ This case is
essentially normal
```

The text string following the "//+" is appended automatically to any HISTORY field text that already exists for the chosen patient and diagnosis. If no HISTORY field text existed, the string would become Line 1 of the HISTORY field text.

When editing the entry, you see the text with the addition and can edit it in the usual way. If you use three slashes ("///") instead of two, the addition is made, and you are not presented with the text to edit.

INPUT Templates

Branching Within INPUT Templates

Sometimes, you want to dynamically control editing based on the responses given for a particular entry or on other aspects of the editing session. By using a technique called branching, the designer of an INPUT template can make the presentation of certain fields conditional based on the values of other fields. You *must* have programmer access to set up branching. With programmer access, any executable M code can be put into an INPUT template.

You can branch either to a field prompt elsewhere in the template or to a predefined place holder. The place holder is identified by **@n**, where "**n**" is an integer (e.g., **@1**).

To branch within an INPUT template, you enter M code at one of the "EDIT FIELD:" prompts. You set the variable Y to the branch destination. Y can be given the value of a field label, a field number, or a place holder. If Y is set to zero and editing is being done at the top level of a file, the template is exited. If Y is set to zero and a Multiple is being edited, the Multiple is exited.

The variable X will contain the *updated*, *internal* value of the field edited at the previous prompt. Thus, you can check X to determine if you want to set Y to branch or not. For example, suppose you had a file called ADMISSIONS. Some of the fields are concerned only with the discharge of a patient. You want to branch around those fields, if the DATE OF DISCHARGE is empty in the database and no date is given in the current editing session. Your template could be defined like this:

```
Select OPTION: ENTER OR EDIT FILE ENTRIES

INPUT TO WHAT FILE: ADMISSIONS
EDIT WHICH FIELD: ALL// NAME
THEN EDIT FIELD: DIAGNOSIS
THEN EDIT FIELD: ADMITTING PHYSICIAN
THEN EDIT FIELD: DATE OF DISCHARGE
THEN EDIT FIELD: S:X="" Y="@1"
THEN EDIT FIELD: DISCHARGING PHYSICIAN
THEN EDIT FIELD: FOLLOW-UP DATE
THEN EDIT FIELD: @1
THEN EDIT FIELD: BILLING METHOD
THEN EDIT FIELD: <Enter>
STORE THESE FIELDS IN TEMPLATE: EDIT ADMISSION
  Are you adding 'EDIT ADMISSION' as a new INPUT TEMPLATE? Y
<Enter>  (YES)
```

This template will branch around the discharge related questions, if the DATE OF DISCHARGE is null.

If you wanted to further enhance the template to ask for MEDICARE NUMBER only if BILLING METHOD is "M" (for Medicare), you could change the template like this:

```
INPUT TO WHAT FILE: ADMISSIONS// <Enter>
EDIT WHICH FIELD: ALL// [EDIT ADMISSION]
     USER #2 FILE #16155
WANT TO EDIT 'EDIT ADMISSION' INPUT TEMPLATE? NO// Y
NAME: EDIT ADMISSION// <Enter>
READ ACCESS: @// <Enter>
WRITE ACCESS: @// <Enter>
```

```
EDIT WHICH FIELD: .01// <Enter>  NAME
THEN EDIT FIELD: 1// <Enter>  DIAGNOSIS
THEN EDIT FIELD: 2// <Enter>  ADMITTING PHYSICIAN
THEN EDIT FIELD: 3// <Enter>  DATE OF DISCHARGE
THEN EDIT FIELD: S:X="" Y="@1"// <Enter>
THEN EDIT FIELD: 4// <Enter>  DISCHARGING PHYSICIAN
THEN EDIT FIELD: 5// <Enter>  FOLLOW-UP DATE
THEN EDIT FIELD: @1// <Enter>
THEN EDIT FIELD: 6// <Enter>  BILLING METHOD
THEN EDIT FIELD: 7// S:X="M" Y="MEDICARE NUMBER"
THEN EDIT FIELD: S Y=0
THEN EDIT FIELD: MEDICARE NUMBER
THEN EDIT FIELD: <Enter>
STORE THESE FIELDS IN TEMPLATE: <Spacebar><Enter>  EDIT
ADMISSION
  (OCT 31, 1991@14:17)      USER #2 FILE #16155
EDIT ADMISSION TEMPLATE ALREADY EXISTS.... OK TO REPLACE? Y
```

After the BILLING METHOD field is edited, a test is made of its contents. It is a DATA TYPE field of SET OF CODES; thus, the test is for the letter "M" alone (the internal value of the field). If it is equal to "M", the template branches to the MEDICARE NUMBER field. If it is not equal to "M", the template proceeds to the next prompt where Y is set unconditionally to zero. The template is exited here so that the "MEDICARE NUMBER" prompt is not shown when it is not needed.

An editing session using this template to add a new admission might look like this:

```
    Select ADMISSIONS NAME: FMPATIENT,19
      Are you adding 'FMPATIENT,19' as a new ADMISSIONS (the
4TH)?  No// Y <Enter>  (Yes)
    DIAGNOSIS: MEASLES
    ADMITTING PHYSICIAN: FMPROVIDER,4
    DATE OF DISCHARGE: <Enter>
    BILLING METHOD: MEDICARE
    MEDICARE NUMBER: 3093-0393
```

The discharge related questions were skipped, and the "MEDICARE NUMBER:" prompt was given. A future editing of this record upon patient discharge could look like this:

```
    Select ADMISSIONS NAME: FMPATIENT,19
             ...OK? YES// <Enter>

    NAME: FMPATIENT,19// ^DATE OF DISCHARGE
    DATE OF DISCHARGE: 5/9/90
    DISCHARGING PHYSICIAN: FMPROVIDER,4
    FOLLOW-UP DATE: 6/1/90
    BILLING METHOD: MEDICARE// <Enter>
    MEDICARE NUMBER: 3093-0393// <Enter>
```

There is a potential hazard in using branching. In this example, suppose the BILLING METHOD were changed to "P" (for private insurance). The simple branching logic used would not show you the MEDICARE NUMBER field to edit or delete. You *must* ensure that your template can handle this kind of situation. In this example, if you have programmer access to do so, you might add M code to delete the MEDICARE NUMBER, if BILLING METHOD were not equal to "M".

Text Formatting in Word Processing Fields

Word Wrapping

Word wrapping is performed when a WORD-PROCESSING-type field is printed. Two functions occur as part of word wrapping during prints: lines are "filled" to the right margin and lines are "broken" only at word breaks.

If word wrap is on (a data dictionary setting for the WORD-PROCESSING-type field in question), you can *override* the word wrapping function and force a line to be printed as it appears in the editor by doing one of the following with the line:

Starting the line with a space.
Pressing the Tab key at the end of the line while using the Line Editor, or type |**Tab**| at the end of the line while using the Screen Editor.
Turning wrap off by using the |**NOWRAP**| function described below.

Lines that contain only punctuation are always printed as is. Thus, if you put a single space on a line, the previous line will not be filled and the subsequent line will begin in column one.

NOTE: The editor's line numbers are meaningful only when editing. Since word processing data is usually printed in a wraparound mode, what is internally line three might be printed as lines five and six.

Tabs

Tabs can be meaningful wherever they occur in a line.

NOTE: If you insert a tab by typing the special Tab key on the keyboard (or **<Ctrl-I>** on terminals without a Tab key), a |**Tab**| will be inserted in the text instead. When editing, a tab is recognized as |**Tab**|, not as five blank spaces.

Formatting Text with Word Processing Windows (Frames) | |

Expressions framed by vertical bars ("| |") are known as word processing *windows* or *frames*, and are evaluated as computed expression at print-time and will be printed as evaluated. (MailMan does not typically evaluate expressions within vertical bars, neither does the Inquire to File Entries option or the CAPTIONED PRINT template.) For example, |**TODAY+1**| will print out tomorrow's date.

You can use word processing windows to insert one of the following into the text of a WORD-PROCESSING-type field when that WORD-PROCESSING-type field is printed:

A Field Name.
A Computed Expression.
Text Formatting Expression.

REF: For details of how to compose and use computed expressions, see the "Computed Expressions" section.

Text Formatting Expressions in Word Processing Windows

The following is the list of recognized special text formatting functions that you can use within word processing windows. Most of these functions can be used in other contexts—for example, at the "PRINT FIELD:" prompt.

Text Formatting Expression	Description
|RIGHT-JUSTIFY|	Causes the text that follows it to be padded with spaces between words, so the right margin is even.
|DOUBLE-SPACE|	Causes the text that follows it to be printed with blank lines inserted every other line.
|SINGLE-SPACE|	Turns off double-spacing for the text that follows it.
|TOP|	Causes a page break to occur at this point.
|NOBLANKLINE|	If nothing will be printed on the line, this causes the line to be suppressed so that a blank line is not output. It is useful if the line contains only a computed expression that might evaluate to null.
|PAGEFEED|(arg)|	Causes page breaks to occur in the text that follows it, whenever fewer than arg number lines remain on the current page.
|PAGESTART|(arg)|	Causes the text on the following pages to begin at line # arg of the page.
|SETPAGE|(arg)|	Resets page numbering, so that the page number that follows it will be arg+1.
|BLANK|(arg)|	Causes arg number of blank lines to be inserted at this point in the text.

Text Formatting Expression	Description
|INDENT|(arg)|	Causes the text that follows it to be indented **arg** number of spaces from the left margin.
|SETTAB|(arg1,arg2,arg3..)|	Sets tab positions for the text that follows it. In subsequent lines, the first **|TAB|** encountered will cause indentation to column position **arg1** characters from the left margin. The second **|TAB|** encountered will cause indentation to column position **arg2**, and so on. If any **SETTAB arg** is negative, the text following the corresponding **|TAB|** will be right justified so that the rightmost column of that text will fall in the column number that is the absolute value of the **SETTAB arg**. If a **SETTAB arg** is the literal "**C**" (i.e., |**SETTAB("C")**|), the text following the corresponding tab setting will be centered.
|CENTER|(arg)|	Causes the **arg** to be centered.
|TAB|	Causes the text to start printing at predetermined indents. The default column settings are 5,10,15,20, ..., which can be reset with **SETTAB**. **|TAB|** at the end of a line causes that line to be printed as is (no word wrapping).

Text Formatting Expression	Description
|TAB n|	Overrides any **SETTAB** specification for the text that follows it and causes tabbing to the **n**th column over from the left margin. Output is right justified on the **n**th column, if "**n**" is negative. For example, the text following **|TAB 12|** will begin at column 12; the text following **|TAB "C"|** will be centered.
|WIDTH|(arg)|	Specifies that the text that follows it will always be printed in a column **arg** characters wide. (**Arg**, in other words, is the difference between the left margin position and the right margin position, plus one.) **NOTE:** In the absence of a **WIDTH** specification, the output column width is determined by the user (or defaulted by the system) at print time.
|NOWRAP|	Causes the text that follows it to be printed line-for-line (without wraparound). This eliminates the need to end each line with a tab or start the line with a space to force the line to be printed as it stands.
|WRAP|	Causes the text that follows it to be printed in wraparound mode. This is the default setting.
|UNDERLINE|(arg)|	Causes the **arg** to be underlined.
|_|	Starts underlining. Underlining continues until a second |_| is encountered. This only works on printers that underline.

REF: For additional information about functions, see the "Computed Expressions" section.

NOTE: In order to print a "|" character, you *must* enter it as "||". Likewise, to print "||" enter "||||".

Glossary

.001 FIELD	A field containing the internal entry number of the record.
.01 FIELD	The one field that *must* be present for every file and file entry. It is also called the NAME field. At a file's creation the .01 field is given the label NAME. This label can be changed.
ALERTS	Brief online notices that are issued to users as they complete a cycle through the menu system. Alerts are designed to provide interactive notification of pending computing activities, such as the need to reorder supplies or review a patient's clinical test results. Along with the alert message is an indication that the View Alerts common option should be chosen to take further action.
ANSI	American National Standards Institute.
ANSI M	The M (formerly known as MUMPS) programming language is a standard recognized by the American National Standard Institute (ANSI). M stands for Massachusetts Utility Multi-programming System.
API	Program calls provided for use by application programmers. APIs allow programmers to carry out standard computing activities without needing to duplicate utilities in their own software. APIs also further DBA goals of system integration by channeling activities, such as adding new users, through a limited number of callable entry points. VISTA APIs fall into

	the following three categories:
	The first category is "Supported API" These are callable routines, which are supported for general use by all VISTA applications.
	The second category is "Controlled Subscription API." These are callable routines for which you *must* obtain an Integration Agreement (IA - formerly referred to as a DBIA) to use.
	The third category is "Private API," where only a single application is granted permission to use an attribute/function of another VISTA package.
	These IAs are granted for special cases, transitional problems between versions, and release coordination.
ARRAY	An arrangement of elements in one or more dimensions. An M array is a set of nodes referenced by subscripts that share the same variable name.
AT-SIGN ("@")	A Fileman security Access Code that gives the user programmer-level access to files and to Fileman's developer features. See Programmer Access. Also, the character "@" (i.e., at-sign) is used at Fileman field prompts to delete data.
AUDIT TRAIL	The record or log of an ongoing audit.
AUDITING	The monitoring and recording of computer use.
BACKWARD POINTER	A pointer to your current file from another file; used in the extended pointer syntax.

BOOLEAN EXPRESSION	A logical comparison between values yielding a true or false result. In M, zero means false and non-zero (often one) means true.
CALLABLE ENTRY POINT	An authorized programmer call that may be used in any VISTA application package. The DBA maintains the list of DBIC-approved entry points.
CANONIC NUMBER	A number with no leading zeros and no trailing zeros after a decimal point.
CAPTION	In Screenman, a label displayed on the screen. Captions often identify fields that are to be edited.
CHECKSUM	The result of a mathematical computation involving the individual characters of a routine or file.
COMMAND AREA	In Screenman, the bottom portion of the screen used to display help information and to accept user commands.
CONTROLLED SUBSCRIPTION INTEGRATION AGREEMENT	This applies where the IA describes attributes/functions that *must* be controlled in their use. The decision to restrict the IA is based on the maturity of the custodian package. Typically, these IAs are created by the requesting package based on their independent examination of the custodian package's features. For the IA to be approved, the custodian grants permission to other VISTA packages to use the attributes/functions of the IA; permission is granted on a one-by-one basis where each is based on a solicitation by the requesting package. An example is the extension of permission to allow a package (e.g., Spinal Cord Dysfunction) to define and update a component that is supported within the Health Summary package file structures.

CROSS-REFERENCE	An attribute of a field or a file that identifies an action that should take place when the value of a field is changed. Often, the action is the placement of the field's value into an index. A Traditional cross-reference is defined with a specific field. A New-Style cross-reference is a file attribute and can be composed of one or more fields. New-Style cross-references are stored in the INDEX file (#.11).
DATA ATTRIBUTE	A characteristic unit of data such as length, value, or method of representation. Fileman field definitions specify data attributes.
DATA DICTIONARY	A record of a file's structure, its elements (fields and their attributes), and relationships to other files. Often abbreviated as DD.
DATA DICTIONARY ACCESS	A user's authorization to write/update/edit the data definition for a computer file. Also known as DD Access.
DATA INTEGRITY	This term refers to the condition of patient records in terms of completeness and correctness. It also refers to the process in which a particular patient's data is synchronized at all the sites in which that patient receives care.
DATABASE MANAGEMENT SYSTEM (DBMS)	A collection of software that handles the storage, retrieval, and updating of records in a database. A Database Management System (DBMS) controls redundancy of records and provides the security, integrity, and data independence of a database.
DATABASE, NATIONAL	A database that contains data collected or entered for all VHA sites.

DBA	Database Administrator, oversees software development with respect to VISTA Standards and Conventions (SAC) such as namespacing. Also, this term refers to the Database Administration function and staff.
DBIA	Database Integration Agreement (see Integration Agreements [IA]).
DICTIONARY	Database of specifications of data and information processing resources. Fileman's database of data dictionaries is stored in the FILE of files (#1).
DIRECT MODE UTILITY	A programmer call that is made when working in direct programmer mode. A direct mode utility is entered at the MUMPS prompt (e.g., >D ^XUP). Calls that are documented as direct mode utilities cannot be used in application software code.
DOMAIN	A site for sending and receiving mail.
DUZ	Local variable holding the user number that identifies the signed-on user.
DUZ(0)	Local variable that holds the File Manager Access Code of the signed-on user.
EDIT WINDOW	In Screenman, the area in which you enter or edit data. It is highlighted with either reverse video or an underline. In Screen Editor, the area in which you enter and edit text; the area between the status bar and the ruler.

ERROR TRAP	A mechanism to capture system errors and record facts about the computing context such as the local symbol table, last global reference, and routine in use. Operating systems provide tools such as the %ER utility. The Kernel provides a generic error trapping mechanism with use of the ^%ZTER global and ^XTER* routines. Errors can be trapped and, when possible, the user is returned to the menu system.
EXTENDED POINTERS	A means to reference fields in files other than your current file.
FIELD NUMBER	The unique number used to identify a field in a file. A field can be referenced by "#" followed by the field number.
FORM	In Screenman, a group of one or more pages that comprise a complete transaction. Comparable to an INPUT template.
FREE TEXT	A DATA TYPE field value that can contain any printable characters.
GLOBAL VARIABLE	Variable that is stored on disk (M usage).
HISTOGRAM	A type of bar graph that indicates frequency of occurrence of particular values.
IDENTIFIER	In Fileman, a field that is defined to aid in identifying an entry in conjunction with the NAME field.

INDEX	An ordered list used to speed retrieval of entries from a file based on a value in some field or fields. The term "simple index" refers to an index that stores the data for a single field; the term "compound index" refers to an index that stores the data for more than one field. Indexes are created and maintained via cross-references.
INPUT TEMPLATE	A pre-defined list of fields that together comprise an editing session.
INTEGRATION AGREEMENTS (IA)	Integration Agreements define agreements between two or more VISTA software applications to allow access to one development domain by another. VISTA software developers are allowed to use internal entry points (APIs) or other software-specific features that are not available to the general programming public. Any software developed for use in the VISTA environment is required to adhere to this standard; as such, it applies to vendor products developed within the boundaries of DBA assigned development domains (e.g., MUMPS AudioFax). An IA defines the attributes and functions that specify access. The DBA maintains and records all IAs in the Integration Agreement database on FORUM. Content can be viewed using the DBA menu or the Health Systems Design & Development's Web page.
INTERNAL ENTRY NUMBER	The number used to identify an entry within a file. Every record has a unique internal entry number. Often abbreviated as IEN.

KERNEL	A VISTA software package that functions as an intermediary between the host operating system and VISTA application packages. Kernel includes installation, menu, security, and device services.
KEY	A group of fields that, taken collectively, uniquely identifies a record in a file or subfile. All fields in a key *must* have values. The term "simple key" refers to keys that are composed of only one field; the term "compound key" refers to keys that are composed of more than one field. Keys are stored in the KEY file (#.31)
LOOKUP	To find an entry in a file using a value for one of its fields.
M (ANSI STANDARD)	Massachusetts General Hospital Utility Multi-Programming System (M, formerly named MUMPS) is a software package, which consists of a high level programming language and a built-in database.
MENU TEXT	The descriptive words that appear when a list of option choices is displayed. Specifically, the Menu Text field of the OPTION file (#19). For example, User's Toolbox is the menu text of the XUSERTOOLS option. The option's synonym is TBOX.
MUMPS	Abbreviated as M. The American National Standards Institute (ANSI) computer language used by Fileman and throughout VISTA. The acronym MUMPS stands for **M**assachusetts General Hospital **U**tility **M**ulti **P**rogramming **S**ystem.

NAME FIELD	The one field that *must* be present for every file and file entry. It is also called the .01 field. At a file's creation the .01 field is given the label NAME. This label can be changed.
NAMESPACE	A convention for naming VISTA package elements. The Database Administrator (DBA) assigns unique character strings for package developers to use in naming routines, options, and other package elements so that packages may coexist. The DBA also assigns a separate range of file numbers to each package.
NODE	In a tree structure, a point at which subordinate items of data originate. An M array element is characterized by a name and a unique subscript. Thus the terms: node, array element, and subscripted variable are synonymous. In a global array, each node might have specific fields or "pieces" reserved for data attributes such as name.
NON-CANONIC NUMBER	A number with either leading zeros, or trailing zeros after a decimal point. M treats non-canonic numbers as text instead of as numbers.
NON-NULL	A value other than null. A space and zero are non-null values.
NULL	Empty. A field or variable that has no value associated with it is null.
NULL RESPONSE	When replying to a prompt, pressing only the **Enter** key, abbreviated as **<Enter>**, to enter nothing.
NUMERIC EXPRESSION	An expression whose value is a number. Compare to string expression.

NUMERIC FIELD	Response that is limited to a restricted number of digits. It can be dollar valued or a decimal figure of specified precision.
OPERATOR	One of the processes done to the elements in an expression to create a value.
OPTION	A computing activity that you can select, usually a choice from a menu.
OPTION NAME	Name field in the OPTION file (e.g., XUMAINT for the option that has the menu text "Menu Management"). Options are namespaced according to VISTA conventions monitored by the DBA.
PACKAGE (SOFTWARE)	The set of programs, files, documentation, help prompts, and installation procedures required for a given application (e.g., Laboratory, Pharmacy, and PIMS). A VISTA software environment is composed of elements specified via the PACKAGE file (#9.4). Elements include files, associated templates, namespaced routines, and namespaced file entries from the OPTION, HELP FRAME, BULLETIN, and FUNCTION files. As public domain software, VISTA software can be requested through the Freedom of Information Act (FOIA).
PATTERN MATCH	In M, an operator that compares the contents of a variable or literal to a specified pattern of characters or kinds of characters.
POINTER TO A FILE	A DATA TYPE field value that contains an explicit reference to an entry in a file. POINTER TO A FILE-type fields are used to relate files to each other.

PREFERRED EDITOR	The editor always entered when you access a WORD-PROCESSING-type field; your default editor. Kernel *must* be present to establish a Preferred Editor.
PRIMARY KEY	A Data Base Management System construct, where one or more fields uniquely define a record (entry) in a file (table). The fields are required to be populated for every record on the file, and are unique, in combination, for every record on the file.
PRINT TEMPLATE	The stored specifications of a printed report, including fields to be printed and formatting instructions.
PRIVATE INTEGRATION AGREEMENT	Where only a single application is granted permission to use an attribute/function of another VISTA package. These IAs are granted for special cases, transitional problems between versions, and release coordination. A Private IA is also created by the requesting package based on their examination of the custodian package's features. Example: one package distributes a patch from another package to ensure smooth installation.
PROGRAMMER ACCESS	The ability to use Fileman features that are reserved for application developers. Referred to as "having the at-sign ('@')" because the at-sign is the DUZ(0) value that grants programmer access.
RECORD NUMBER	See Internal Entry Number.
RELATIONAL NAVIGATION	Changing your current (or primary) file reference to another file. Relational navigation is accomplished by using the extended pointer syntax without specifying a field in the referenced file.

ROUTINE	Program or a sequence of instructions called by a program that may have some general or frequent use. M routines are groups of program lines, which are saved, loaded, and called as a single unit via a specific name.
SAC	Standards and Conventions. Through a process of quality assurance, all VISTA software is reviewed with respect to SAC guidelines as set forth by the Standards and Conventions Committee (SACC).
SACC	VISTA's Standards and Conventions Committee. This Committee is responsible for maintaining the SAC.
SCATTERGRAM	A graph in which occurrences of two fields are displayed on an X-Y coordinate grid to aid in data analysis.
SEARCH TEMPLATE	The saved results of a search operation. Usually, the actual entries found are stored in addition to the criteria used to select those entries.
SECURITY KEY	The purpose of Security Keys is to set a layer of protection on the range of computing capabilities available with a particular software package. The availability of options is based on the level of system access granted to each user.

SENSITIVE PATIENT	Patient whose record contains certain information, which may be deemed sensitive by a facility, such as political figures, employees, patients with a particular eligibility or medical condition. If a shared patient is flagged as sensitive at one of the treating sites, a bulletin is sent to the DG SENSITIVITY mail group at each subscribing site telling where, when, and by whom the flag was set. Each site can then review whether the circumstances meet the local criteria for sensitivity flagging.
SEPG	Software Engineering Process Group (SEPG) (renamed the Engineering Process Group [EPG])
SET OF CODES	A DATA TYPE field value where a short character string is defined to represent a longer value.
SIMPLE EXTENDED POINTERS	An extended pointer that uses a pre-existing pointer relationship to access entries in another file.
SOFTWARE (PACKAGE)	The set of programs, files, documentation, help prompts, and installation procedures required for a given application (e.g., Laboratory, Pharmacy, and PIMS). A VISTA software environment is composed of elements specified via the PACKAGE file (#9.4). Elements include files, associated templates, namespaced routines, and namespaced file entries from the OPTION, HELP FRAME, BULLETIN, and FUNCTION files. As public domain software, VISTA software can be requested through the Freedom of Information Act (FOIA).

SORT TEMPLATE	The stored record of sort specifications. It contains sorting order as well as restrictions on the selection of entries. Used to prepare entries for printing.
SPACEBAR RETURN	You can answer a Fileman prompt by pressing the spacebar and then the Return key. This indicates to Fileman that you would like the last response you were working on at that prompt recalled.
SPECIAL QUEUING	Option attribute indicating that Task Manager should automatically run the option whenever the system reboots.
STUFF	To place values directly into a field, usually with no user interaction.
SUBENTRY	An entry in a Multiple; also called a Subrecord.
SUBFIELD	A field in a Multiple.
SUBFILE	The data structure of a Multiple. In many respects, a Subfile has the same characteristics as a File.
SUBSCRIPT	A symbol that is associated with the name of a set to identify a particular subset or element. In M, a numeric or string value that: is enclosed in parentheses, is appended to the name of a local or global variable, and identifies a specific node within an array.

SUPPORTED REFERENCE INTEGRATION AGREEMENT	This applies where any VISTA application may use the attributes/functions defined by the IA (these are also called "Public "). An example is an IA that describes a standard API such as DIE or VADPT. The package that creates/maintains the Supported Reference *must* ensure it is recorded as a Supported Reference in the IA database. There is no need for other VISTA packages to request an IA to use these references; they are open to all by default.
TASK MANAGER	Kernel module that schedules and processes background tasks (also called TaskMan)
TEMPLATE	Means of storing report formats, data entry formats, and sorted entry sequences. A template is a permanent place to store selected fields for use at a later time. Edit sequences are stored in the INPUT TEMPLATE file (#.402), print specifications are stored in the PRINT TEMPLATE file (#.4), and search or sort specifications are stored in the SORT TEMPLATE file (#.401).
TRIGGER	A type of Fileman cross-reference. Often used to update values in the database given certain conditions (as specified in the trigger logic). For example, whenever an entry is made in a file, a trigger could automatically enter the current date into another field holding the creation date.
UCI	User Class Identification, a computing area. The MGR UCI is typically the Manager's account, while VAH or ROU may be Production accounts.
UPLOAD	Send a file from one computer system to another (usually using communications software).

USER ACCESS	This term is used to refer to a limited level of access, to a computer system, which is sufficient for using/operating a package, but does not allow programming, modification to data dictionaries, or other operations that require programmer access. Any option, for example, can be locked with the key XUPROGMODE, which means that invoking that option requires programmer access. The user's access level determines the degree of computer use and the types of computer programs available. The System Manager assigns the user an access level.
VA	Department of Veterans Affairs
VARIABLE	Character, or group of characters, that refer(s) to a value. M (previously referred to as MUMPS) recognizes 3 types of variables: local variables, global variables, and special variables. Local variables exist in a partition of main memory and disappear at sign-off. A global variable is stored on disk, potentially available to any user. Global variables usually exist as parts of global arrays. The term "global" may refer either to a global variable or a global array. A special variable is defined by systems operations (e.g., $TEST).
VHA	Veterans Health Administration.
VISN	Veterans Integrated Service Network
VPID	Veterans Administration Personal Identifier.

Index